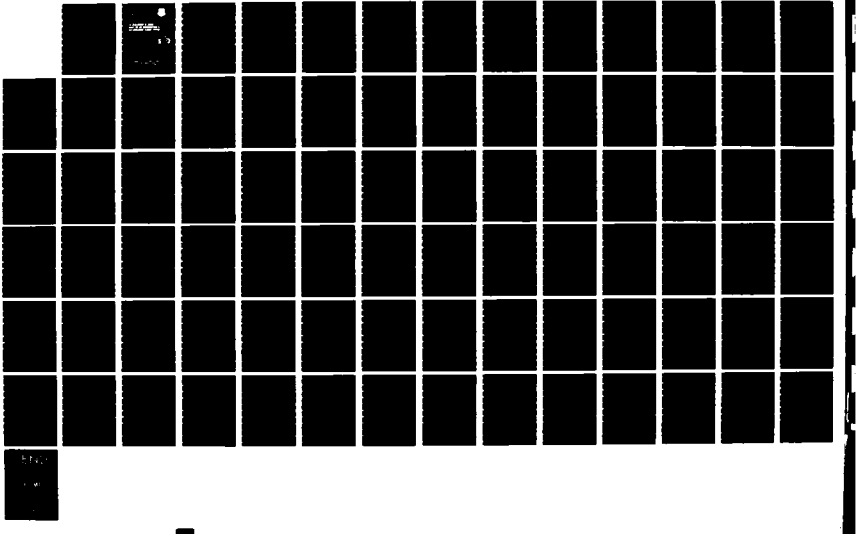
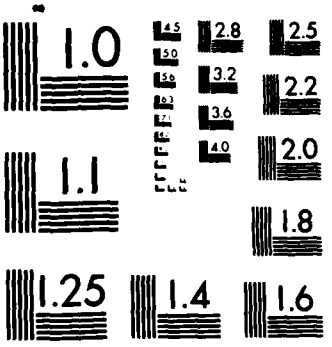
AD-A163 055 THE DEVELOPMENT OF DESIGN GUIDES FOR THE IMPLEMENTATION 1/1
OF MULTIPROCESSIN (U) UNIVERSITY OF MANCHESTER INST OF
SCIENCE AND TECHNOLOGY (ENGL D ASPINALL SEP 85
UNCLASSIFIED UMIST/RADC/1 RADC-TR-84-273 F/G 9/2 NL





RADC-TR-84-273
Final Technical Report
September 1985

THE DEVELOPMENT OF DESIGN GUIDES FOR THE IMPLEMENTATION OF MULTIPROCESSING ELEMENT SYSTEMS

The University of Manchester

Professor D. Aspinall

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-84-273 has been reviewed and is approved for publication.

APPROVED: Frederick A. Normand

FREDERICK A. NORMAND
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Command and Control Division

FOR THE COMMANDER:

DONALD A. BRANTINGHAM
Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTC) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A163 055

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UMIST/RADC/1	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-84-273
6a. NAME OF PERFORMING ORGANIZATION The University of Manchester 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COTC)
6c. ADDRESS (City, State, and ZIP Code) Institute of Science and Technology PO Box 88 Manchester M60 IQD England	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) COTC	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F49620-80-C-0012
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62702F; PROJECT NO: 5581; TASK NO: 17; WORK UNIT ACCESSION NO: 16

11. TITLE (Include Security Classification)
THE DEVELOPMENT OF DESIGN GUIDES FOR THE IMPLEMENTATION OF MULTIPROCESSING ELEMENT SYSTEMS

12. PERSONAL AUTHOR(S)
Professor D. Aspinall

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 1980 TO 1984	September 1985	88

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Distributed Systems Real-Time Systems Multi-Microprocessor Interconnections
09	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Multi Processing-Element Systems are complex and it is important that designers have as much help as possible in arriving at an efficient and reliable design. The primary objective of the research project was to identify aspects of a design which could be investigated through simulation. Two aspects of the problem were identified. The first of these was the interconnection structure or system by which the separate processing elements are interconnected. The second concerned the language to be used in order to design the processes to be distributed throughout the Multi Processing-Element Systems.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Frederick A. Normand	22b. TELEPHONE (Include Area Code) (315) 330-2925 22c. OFFICE SYMBOL RADC (COTC)

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

CONTENTS

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

The prospect of low cost computing components provides the designer of complex information processing systems with the possibility of using many processing elements operating concurrently. Such Multi Processing-element Systems are clearly going to be complex and it is important that the designers of such systems have as much help as possible in arriving at an efficient and reliable design. The primary object of the research project was to attempt to identify aspects of the design which could be investigated on a particular example of a Multi Processing-element System. Two aspects of the problem were identified. The first of these was the interconnection structure or system by which the separate processing elements are interconnected. The second concerned the language to be used in order to design the processes to be distributed throughout the Multi Processing-element System.

1.1 INTERCONNECTION STRUCTURE

Several alternative interconnection structures have been identified and placed in a taxonomy by Anderson and Jensen. An example of a direct shared memory exists in the Department of Computation at UMIST as a research vehicle. A particular feature of this system is a collection of conventional microcomputers connected through a direct shared memory of high performance. Measurements of the performance of this system (1) have indicated that there is very little degradation in performance as the number of active processing elements increases. It is therefore possible to undertake simulation exercises on this structure which simulate different interconnection structures in a realistic manner. Initial experiments were carried out to identify certain components of interconnection structures which could be simulated effectively in this direct shared memory system. These were found to be cumbersome in use and it was decided to abandon this general purpose approach and concentrate upon the development of one particular interconnection structure on the direct shared memory and to investigate this one rigorously. Of all the various interconnection structures that were possible, it was decided to concentrate upon the token passing ring system. An extensive model to simulate a unidirectional ring has been developed and used in several experiments. Through these experiments it has been possible to extract certain important parameters which characterise the performance of a token passing ring. These experiments lead to the development of theoretical models which enable the analysis of the performance of such rings.

1.2 LANGUAGE IMPLICATIONS

The design of Multi Processing-element Systems will depend upon the quality of real time language to be used. Several real time languages have been developed recently and it is interesting to consider the interaction between the structure of Multi Processing-element Systems and the execution of programs written in these languages. A detailed investigation of the CHILL language has indicated that there are likely to be serious problems in the implementation of certain central features on distributed systems. Some features of these languages suggest that a ring structure would be perfectly adequate for their needs, while other features seem to demand a shared memory approach. These conflicting requirements will have an effect upon the performance of a complete Multi Processing-element. An important part of the investigation has been

to consider the primitives of such real time languages and the implementation of these languages upon Multi Processing-element Systems.

1.3 EXPERIMENTAL SYSTEM

Investigations have been carried out upon a Multi Processing-element System based upon a direct shared memory. An important feature of this system has been the existence of a comprehensive development environment which has made it possible to develop the programs necessary for the investigation. At one stage of the project, it was felt necessary to improve the performance of certain nodes of the Multi Processing-element Systems in order to extend the scope of the investigation. Attempts have been made to achieve this by using higher performance microprocessors and by developing a special node based upon the "bit slice".

Towards the end of the project, the value of the multiprocessor simulation tool and the theoretical models for the token passing ring were identified. It is believed that this preliminary investigation of the design guide for the implementation of Multi Processing-element Systems has produced some important metrics for the characterisation of such systems and it is now possible to continue development of these design guides through the use of simulation techniques on a conventional uniprocessor system.

Furthermore, the limitation of the processors within the system and their inefficient support of high level languages were appreciated. In parallel with the experiments on the Multi Processing-element Systems simulation models were constructed in a high performance uniprocessor. This experiment has suggested that it is necessary to gain an understanding of the critical parameters by experiments with a system which includes genuine, rather than simulated, concurrency, but once this understanding can be expressed in a consistent set of parameters, then the convenience of a high performance uniprocessor supporting a concurrent high level language has much more to commend it, as a vehicle for simulation experiments.

2. DEVELOPMENT OF A SIMULATION AND THEORETICAL MODEL FOR A TOKEN PASSING RING

2.1 INTRODUCTION

The modelling of multiprocessor interconnection structures on a Direct Shared Memory (DSM) multi-microprocessor has formed a large part of the work performed on this contract. One such interconnection structure is the "ring", where the processing elements are connected in a circular arrangement. A general ring simulation model has been developed and applied to the analysis of a "token passing ring". In addition, a theoretical ring performance model has been derived for predicting the performance characteristics of token passing rings under varying traffic flow pattern conditions. The model has proved to be of value in the development of guidelines for predicting the operational characteristics of ring structures. The theory developed has been applied to the analysis of unidirectional and bidirectional ring structures.

An overview of the ring simulation model is presented, together with the derivation of the performance model. Finally, preliminary results are presented which indicate the close correlation between results obtained from the simulation model and the theoretical model.

2.2 RING SIMULATION MODEL

Many ring simulation models have been proposed in the literature (2, 3, 4), which are implemented on single processor machines with limited resources, to model the inherent parallelism present in a ring structure. The simulation model, developed at UMIST, overcomes these deficiencies by implementing the model on the shared memory multiprocessor, CYBA-M (5). The ring nodes map directly onto processing elements and the ring connection structure can be emulated in the shared memory.

CYBA-M consists of 15 microprocessors (18080s) which communicate through a shared "global memory" of 16K bytes. Each microprocessor has a dedicated local, private memory of 32K bytes. The microprocessor memory pair is known as a "processing element". The 16th processing element is a PDP 11/34, which is used to load programs and data into the different memory areas of CYBA-M. The PDP 11/34 executes an "operating system" known as EMU (6), which provides run-time operating and program debugging facilities for CYBA-M.

The ring simulation model uses a pair of processing elements to model the operation of a "node" in a token passing ring. One of the processing elements acts as "host processor" and is used for user application processes. The other processing element acts as a "communications" processor, which interfaces the host processor to the ring. The ring interconnection structure is modelled by data structures within the global memory, see Appendix A.

The first version of the ring simulation model emulates a unidirectional token passing ring, which uses the HDLC message transmission format (7). A message generated by the host processor enters the output queue of the communications processor. The output queue acts as FIFO buffer. The communications processor will then encapsulate the message in the HDLC format and wait for the reception of the "token" before transmitting the message, one byte at a time, onto the ring. When the token is received, the message is streamed

onto the ring and the token is appended. The message is relayed by the intervening nodes to its destination. The destination communications processor receives the message, strips off the HDLC format and passes it to the host processor. The message is also removed from the ring. If a node receives the token and its output queue is empty, the token passes to the next node having experienced a one byte delay. Further details of the message transmission protocol are given in Appendix A.

2.2.1 Simulation Model Features

The operational characteristics of the ring can be defined interactively for any simulation run. The following parameters are defined prior to a simulation experiment.

(a) The number of nodes present in the ring. The maximum number of nodes is seven, due to the limitation of the number of processing elements in CYBA-M.

(b) The depth of the output queue for the communications processors.

(c) The state of each node: active or inactive. An active node generates messages which are streamed onto the ring. An inactive node does not transmit messages onto the ring.

(d) The length of the message to be transmitted by each active node. The message length is fixed for the duration of a simulation run. The HDLC protocol encapsulation adds five bytes to the length of a "message packet".

(e) The destination node addresses of messages transmitted by an active node. The address may be fixed or randomised.

(f) The number of messages to be transmitted by an active node during a simulation run.

(g) The message transmission mode: constant or exponential. In the constant transmission mode, the message transmission rate is constant. In the exponential mode, the message transmission rate conforms to a Poisson distribution centred on a specified mean.

The operation of the simulation model is tied to a rigidly defined clock and there is a one byte movement around the ring in every simulation clock cycle. Thus, the simulation time can be normalised to operate with a ring of any raw data transmission rate; for example, if the ring speed was 10 Mbts/s, then a simulation clock cycle would represent 800 ns of real time. The simulation model provides the following statistics for every message transmitted by an active node:

(a) Message source, message destination and message number.

(b) Message Send Request Time (SR): This is the time, in terms of simulation clock periods, when a host inserts a message into the communications processor output queue.

(c) Ring Accept Time (RA): This is the time, in terms of simulation clock periods, when the communications processor streams the first byte of a message onto the ring.

(d) Ring Remove Time (RR): This is the time, in terms of simulation clock periods, when the first byte of a message arrives at the communications processor of the message destination node.

(e) Destination Accept Time (DA): This is the time, in terms of simulation clock periods, when the last byte of a message is received by the destination host processor.

The initial state of the simulation model is fixed for every simulation run with the token (one byte long) resident in node 0 at simulation time = 0.

Before proceeding with an analysis of the performance of a token passing ring under varying traffic pattern condtions, we will develop the theoretical (analytical) model of the ring, based on our simulation model.

2.3 THEORETICAL MODEL FOR A TOKEN PASSING RING

The theoretical model establishes stability criteria for token passing rings, and determines the saturation state characteristics for unstable systems. A ring is said to be stable if the operation of the ring remains "steady" in time; that is, a stable system operates in a "normal" state without message congestion and output queue overflows. The saturation state is referred to as the state where the ring is fully utilised and the rate of message transports around the ring cannot be increased.

The model is based on a number of formal parameters:

(a) Number of nodes in the ring (N).

(b) Length of output queue (Q):
Each communications processor has the same length output queue.

(c) Average node packet length (S_i):

The average length of each message generated by node i, including the HDLC headers. If a node is inactive then $S_i = 0$.

(d) Average message generation period (D_i):

The average period between two consecutive messages generated by the host processor of node i. Hence $1/D_i$ represents the average message injection rate into the output queue of node i.

(e) Scan Time (ST):
The scan time is defined as the time taken for the token byte to complete one cycle of the ring. The minimum scan time for the token is achieved when no messages are transmitted onto the ring. As the token is delayed for one simulation clock period at each node, then the minimum scan time is N simulation clock periods for an N node ring.

(f) Mean Scan Time (MST):
This represents the mean scan time for the token. The duration of the scan time will vary according to the message traffi . generated by each node. However, at a constant message sending rate for each node, the ring scan times are periodic and it is possible to define a mean scan time. The derivation of the mean scan time is discussed later.

(g) Node Utilisation Factor (U_i):
The node utilisation factor is defined as S_i/D_i for node i, that is, the time taken to transmit the message from a node. Hence, the utilisation factor represents the ratio at which work is generated by a node to the capacity of a node for performing work. Therefore, for a node to operate correctly U_i must not exceed unity. U_i also represents the fraction of the time node i is busy transmitting messages.

(h) Ring Utilisation Factor (R):
The ring utilisation factor represents the capacity of the ring for conveying messages. It is similar to U_i, and is defined as the fraction of the time the ring is busy servicing the messages generated by the nodes.

The ring allocation strategy is similar to polling, because at any one instant only one node has access to the ring. Each node is, therefore, polled by the token in a round-robin fashion. The amount of work generated by the nodes is equal to the sum of the work generated by the individual nodes; that is, the sum of the individual node utilisation factors.

However, there is an additional overhead due to the presence of the token and the ring utilisation factor must contain a term to account for the fraction of the time spent in circulating the token in the ring. It takes N simulation clock periods for the token to complete one cycle of an N node ring, with no messages being generated. Therefore, we define D_j, such that

$$\text{For all } D_i,\ 0 \leq i \leq N-1,\ i \neq j : \frac{1}{D_j} \leq \frac{1}{D_i}$$

Node j is the fastest message generator in the ring. As each message must be followed by the token, the frequency of the token is governed by the fastest message generating node, and is equal to $1/D_j$. The fraction of time spent in transporting the token around the ring is, therefore, N/D_j. The ring utilisation factor is

$$R = \left(\frac{N}{D_j}\right) + \sum_{i=0}^{i=N-1} U_i \qquad \ldots (2.1)$$

For all Di, $0 \leq i \leq N-1$, $i \neq j$: $\frac{1}{D_j} \leq \frac{1}{D_i}$

2.3.1 Stable Systems

For a ring to maintain operation in the "unsaturated" region, the ring utilisation factor must not exceed unity. Therefore, the condition for ring stability is:

$$\left(\frac{N}{D_j}\right) + \sum_{i=0}^{i=N-1} U_i \leq 1 \qquad \ldots (2.2)$$

For all D_i, $0 \leq i \leq N-1$, $i \neq j$: $\frac{1}{D_j} < \frac{1}{D_i}$

The above condition holds for every ring structure and the case of no message senders, the frequency of the token $(1/D_j)$ is $1/N$, and hence $R = 1$.

The above condition can be rewritten as:

$$\frac{(N + S_j)}{(D_j)} + \sum_{\substack{i=0 \\ i \neq j}}^{i=N-1} U_i \leq 1 \qquad \ldots (2.3)$$

Consider the case, $R = 1$, which only occurs when the ring is fully utilised either by the token alone or, more generally, when the message injection rate into the output queue of node j is equal to the rate of transmission of messages from node j. This represents the steady state of message transport around the ring, and, therefore, D_j must represent the mean scan time of the token.

Therefore,

$$\text{when } R = 1,\ D_j = MST \qquad \ldots (2.4)$$

Substituting for D_j in equation 2.3, we obtain a value for the mean scan time of stable systems when R = 1:

$$MST = \frac{N + S_j}{1 - \sum_{\substack{i = 0 \\ i \neq j}}^{i = N-1} U_i} \qquad \ldots (2.5)$$

For all D_i, $0 \leq i \leq N - 1$, $i \neq j$: $\frac{1}{D_j} > \frac{1}{D_i}$

2.3.2 Unstable Systems

For an unstable system, the main parameter to be determined is the ring saturation time; that is, when the output queue of a node overflows. If the system is unstable then saturation is achieved when the output queue of the fastest node overflows; that is when $1/D_j < 1/MST$. In this case, the rate of injection of messages into the output queue is greater than their rate of departure when the ring is fully utilised (R = 1). Therefore, after a time, Tsat, the output queue overflows. We can apply simple queuing theory (7) to determine the saturation time of the fastest sender in the ring.

Let Min (t) and Mout (t) represent the average number of arrivals and departures of messages from a node in the interval (0, t), respectively. It can be shown (8) that

$$\text{Min (Tsat)} = \text{trunc} \left[\frac{Tsat}{D_j}\right] \qquad \ldots (2.6)$$

$$\text{Mout (Tsat)} = \text{trunc} \left[\frac{Tsat}{MST}\right] - 1 \qquad \ldots (2.7)$$

$$Q = \text{Min (Tsat)} - \text{Mout (Tsat)} \qquad \ldots (2.8)$$

Thus, it is possible to predict the output queue length required for a node, given a specified saturation time.

It is also possible to develop simulation and theoretical models for a bidirectional token passing ring. The same basic premises apply to a bidirectional ring and a preliminary analysis of this ring type is given in Appendix B.

2.3.3 Comparison of the results produced by the Simulation and Theoretical Models

A number of simulation runs have been performed to assess the correlation of the results produced by the simulation model and the theoretical model. In the majority of the simulation runs, the chosen ring configurations were operated close to their saturation points; that is, with R and approximately 1. It should be noted that throughout the analysis the destination of a message plays no part in the stability of a token passing ring. However, the location of the "fastest" message sending node does produce slight perturbations in the saturation time of the ring : this is discussed below.

A major problem in the simulation model is the determination of the mean scan time for the token. One of two methods have been adopted in the calculation of the MST in the theoretical model:

(a) The most convenient method for calculating the MST of a ring is to allow the fastest node to transmit a large number of messages,say k messages, and obtain a value for MST by

$$MST = \frac{RA_k}{k} \qquad \ldots\ldots (2.9)$$

where RA_k is the time when the k'th message has been accepted onto the ring.

In the case of unstable systems, the value of k is limited and it is the number of the last message delivered onto the ring before the node saturated. This method is extremely simple and is an effective way to obtain a value for MST. However, the choice of k is critical as it must be large enough to allow the ring to reach a 'steady state'.

(b) It has been noticed (10) that in certain cases, the token scan times form a repetitive pattern after n scans. If we allow a certain number of messages to be transmitted onto the ring, say m messages, and monitor the ring accept times for the next n messages, then the MST can be obtained from the following formula :

$$MST = \frac{\sum_{i=m}^{i=m+n-1} (RA_{i+1} - RA_i)}{n} \qquad \ldots\ldots (2.10)$$

This method is more accurate and relies on the ring utlisation factors of all the nodes, except the fastest node, to be "exact" fractions; otherwise, no repetitive scan pattern emerges.

An additional problem is due to 'start-up' transients produced by the ring simulation model before it reaches a steady state. This is due to the token always being at node 0 at the start of a simulation run. Hence, the position of the fastest sender, relative to node 0, affects the time of the first ring accept which leads to variations in the ring saturation time, Tsat. However, this does not affect the validity of the theoretical model.

Two examples will be used to illustrate the correlation between the simulation and the theoretical model.

(a) EXAMPLE 1

The ring consists of three nodes with the following characteristics:

Node (i)	S_i (bytes)	D_i (simulation clock period)	U_i (bytes per simulation period)
0	6	18	6/18
1	6	15	6/25
2	6	25	6/25

The ring utilisation factor can be obtained from equation 2.1. Note all nodes have a constant sending rate:

$$R = \frac{6}{18} + \frac{6}{25} + \frac{6}{25} + \frac{3}{18} = 0.98$$

Therefore, the ring is stable ($R < 1$), which was confirmed by a long simulation run. The mean scan time can be predicted from equation 2.5.

$$MST = \frac{3 + 6}{1 - \left(\frac{6}{25} + \frac{6}{25}\right)} = 17.31$$

The MST of the token may be obtained from method (b) as both U_1 and U_2 are exact fractions and the token scan time repeated every 13 scans. It was found that after 13 messages:

$$MST = \frac{225}{13} = 17.31$$

If the average message generation period (D_i) for node 0 is changed from 18 to 17,
then from equation 2.1:

$$R = \frac{6}{17} + \frac{6}{25} + \frac{6}{25} + \frac{3}{17} = 1.01$$

Therefore, the ring is unstable and the simulation model indicated that the ring saturation time, Tsat = 2363. By varying the position of the fastest sender node in the ring, Tsat will vary (for the reasons outlined above). The experiment was repeated with the position of the fastest node varied in each run. The results are given below:

Fastest Node position	Tsat	Min(Tsat)		Mout(Tsat)	
		predicted	actual	predicted	actual
0	2363	139	139	135	135
1	2262	133	133	129	129
2	2364	139	139	135	135

In all cases, the length of the output queue for each node (Q) was equal to four messages, which is confirmed by the predicted and actual values.

(b) EXAMPLE 2

The ring consists of four nodes with the following characteristics:-

Node (i)	S_i	D_i	U_i
0	7	43	7/43
1	9	26	9/26
2	6	50	6/50
3	10	50	10/50

The ring utilisation factor, from equation 2.1, is:

$$R = \frac{7}{43} + \frac{9}{26} + \frac{6}{50} + \frac{10}{50} + \frac{4}{26} = 0.98$$

Therefore, the ring was stable, which was confirmed by a long simulation run. The mean scan time can be predicted from equation 2.5:

$$MST = \frac{4 + 9}{1 - \left(\frac{7}{43} + \frac{6}{50} + \frac{10}{50}\right)} = 25.13$$

The MST of the token may be obtained from method (a) as U_o is not an exact fraction. The simulation results indicated that the 350'th message (the last one for the simulation run) was accepted onto the ring at time, $RA_{349} = 8790$.
Hence, by equation 2.9:

$$MST = \frac{8790}{350} = 25.11$$

If the average message generation period (D_i) for node 1, is changed from 26 to 25, then from equation 2.1:

$$R = \frac{7}{43} + \frac{9}{25} + \frac{6}{50} + \frac{10}{50} + \frac{4}{25} = 1.003$$

Therefore, the ring is unstable and the simulation model indicated that the ring saturation time, Tsat = 8826. Again, by varying the position of the fastest sender node in the ring, Tsat will vary. The experiment was repeated with the position of the fastest node varied in each run. The results are given below:

Fastest Node position	Tsat	Min(Tsat)		Mout(Tsat)	
		predicted	actual	predicted	actual
0	9475	379	379	375	375
1	8826	353	353	349	349
2	10776	431	431	427	427
3	10126	405	405	401	401

In all cases, the length of the output queue for each node (Q) was equal to four messages, which is confirmed by the actual and predicted values.

2.4 CONCLUSIONS

A general simulation model has been developed for analysing the characteristics of ring interconnection structures. The simulator is executed on the multi-microprocessor CYBA-M. The current version of the simulation model has been applied to the study of a token passing ring. In addition, a theoretical model has been derived for predicting the performance of the same ring structure. The definition of the theoretical model is consistent with the simulation model; however, they are independent and the theoretical model is not limited by the constraints imposed by the simulation model. The theoretical model has been developed through queuing analysis and it may easily be extended to cover other ring message passing protocols. In fact, Hayes and Sherman (11) adopted a similar approach in their analysis of the Pierce Ring, whose operation is based on the asynchronous multiplexing of fixed size message packets onto a ring.

Preliminary results indicate a close correlation between the two models and additional experiments performed on the simulation model are presented in the next chapter.

3. PERFORMANCE ANALYSIS OF A TOKEN PASSING RING

3.1. INTRODUCTION

A stability theory has been developed for the analysis of unidirectional (and bidirectional) token passing rings. Early experiments have proved the validity of the stability theory. It is now necessary to extend these experiments in order to evaluate the general performance characteristics, whilst applying the stability theory, where appropriate, to predict their behaviour pattern.

A set of performance metrics are defined, which permit "standard" ring measurements to be performed in order to investigate the effects of varying different ring parameters. Subsequently, a set of standard performance curves are defined, together with the variables that affect the performance characteristics of a token passing ring (13). Finally, a set of actual performance measurements are given which indicate the features of a token passing ring.

3.2 PERFORMANCE METRICS

There are three basic performance measures for a ring: throughput demanded, throughput carried and message delay. The measures need to be determined on an individual node basis and for the ring as a whole.

3.2.1 Throughput Demanded

Throughput demanded is a measure of the required data transfer rate for a node, in terms of bits/s. It is defined as the ratio of the packet length to be transmitted by a node to the rate at which the messages are to be transmitted. Therefore, the throughput demanded for node i is:

$$TD_i = \frac{S_i}{D_i} \qquad 0 \leq i \leq N - 1 \qquad \ldots\ldots (3.1)$$

Thus, TD_i is the same as the "node utilisation factor" defined in the theoretical model.

The throughput demanded for the complete ring; that is, all N ring nodes is defined as:

$$RD = \sum_{i=0}^{i=N-1} TD_i \qquad 0 \leq i \leq N - 1 \ldots\ldots (3.2)$$

Note that RD does not include the token overhead, as this is the function of the ring protocol only.

3.2.2 Throughput Carried

Throughput carried is a measure of the data transfer rate achieved by a ring node, in terms of bits/s. The throughput carried by a node may be measured in one of three ways:

METHOD 1:

By allowing a fixed output queue length for each node, the throughput carried by each node is defined as:

$$TC_i = \frac{S_i . M_i}{Tsat} \qquad 0 \leq i \leq N - 1 \qquad \ldots\ldots (3.3)$$

M_i represents the number of messages generated by node i which are delivered to their destination nodes during the working period of the ring; that is, from start time to the time when the ring saturates, Tsat. The saturation time refers to the time when at least one node output queue overflows.

METHOD 2:

In this case each node is provided with output queues which are long enough to prevent queue overflows. The necessary throughput carried measurements are, thus, undertaken whilst the ring is in operation. The throughput carried by any node can be obtained at any arbitrary time, Ts: the snapshot time. A snapshot of a node is taken at time Ts and the number of messages present in each output queue is obtained. The throughput carried for node i is defined as:

$$TC_i = \frac{m . S_i}{DA_{i,m}} \qquad \ldots\ldots (3.4)$$

$$0 \leq i \leq N - 1$$

$$SR_{i,m} \leq Ts \leq SR_{i,m+1}$$

Where m represents the message number and $DA_{i,m}$ is the time when the m'th message of node i is accepted by the destination host. SR_i, represents the time of entry of the m'th message into the output queue of node i, which is the last message to be entered before the snapshot is taken. The throughput carried is determined by the 'throughput' of the last message to be entered into the output queue.

METHOD 3:

In this case, which is similar to Method 2, the throughput carried for a node is obtained by measuring the throughput of all the messages which have been transmitted up to the snapshot time, and then calculating the average throughput of the node:

$$TC_i = \frac{S_i}{m} \sum_{j=1}^{j=m} \frac{j}{DA_{i,j}} \qquad \ldots\ldots (3.5)$$

$$SR_{i,} \leq Ts \leq SR_{i,m+1}$$

The throughput carried by the ring, that is, all N nodes, is defined as:

$$Rc = \sum_{i=0}^{i=N-1} TC_i \qquad 0 \leq i \leq N-1 \qquad .....(3.6)$$

3.2.3 Message Delay

Message delay is defined as the elapsed time between the generation of a message by host i and its reception by host j. Therefore, for message j of node i, the message delay is defined as:

$$MD_{j,i} = DA_{j,i} - SR_{j,i} \qquad ..... (3.7)$$

The total message delay for node i can be obtained by taking a snapshot of the node at time Ts and calculating the delay from:

$$MD_i = \frac{1}{S_i} \sum_{j=1}^{j=m} MD_{j,i} \qquad ..... (3.8)$$

$$0 \leq i \leq N-1$$

$$SR_{i,m} \leq Ts \leq SR_{i,m+1}$$

The delay is normalised with respect to the message transmission time (S_i), and can be compared to the 'transfer delay' performance metric as defined by Bux (14).

The total message delay for the delay ring, that is all N nodes, is defined as:

$$MD = \sum_{i=0}^{i=N-1} MD_i \qquad 0 \leq i \leq N-1 \qquad ..... (3.9)$$

An expression for MD_i can be derived from the theoretical ring model, in terms of the ring parameters. If the MST > D_i, then message delays start to increase as a queue of messages is formed. The delays form an arithmetic progression with a common factor of (MST - D_i). Hence the total access delay for m messages is:

$$\text{Access Delay for m messages} = \frac{m\,(m+1)\quad(MST - D_i)}{2} \qquad ..... (3.10)$$

When a message is present on the ring it travels an average distance of N/2 and the time taken by the destination host to absorb the message is $(S_i - 2)$. Hence,

$$MD_i \underset{(MST > D_i)}{(\text{theoretical})} = \frac{m}{S_i}\left[\frac{(m + 1)\ (MST - D_i)}{2} + \frac{N}{2} + (S_i - 2)\right] \quad \ldots\ldots \quad (3.11)$$

Equation (3.11) is only valid for $MST > D_i$. If $MST < D_i$ then the access delays remain constant, on average, at MST/2. Hence we have:

$$MD_i \underset{(MST < D_i)}{(\text{theoretical})} = \frac{m}{S_i}\left[\frac{MST}{2} + \frac{N}{2} + (S_i - 2)\right] \quad \ldots\ldots \quad (3.12)$$

3.3 PERFORMANCE CURVES

In order to compare the performance of different token passing ring configurations it is necessary to define a set of standard performance curves. The performance curves are chosen to highlight the major operational characteristics of the ring. Three such curves have been identified:

(a) Throughput carried by the ring (Rc) versus the throughput demand from the ring (RD).

Curves may be produced using the three methods chosen to measure the throughput carried by a node, TC_i. TC_i can be calculated easily from method 1; however, the method assumes that the ring saturation time is deterministic at any message demand rate. The second method is independent of the ring saturation time, but is dependent on the snapshot time Ts. The third method is also dependent on the snapshot time, but overcomes the possible synchronisation effects introduced by the second method.

(b) Total message delay (MD) versus the throughput demanded from the ring (RD).

(c) Ring saturation time (Tsat) versus the throughput demanded from the ring (RD).

This curve indicates how fast a ring will saturate as the throughput demanded is increased. It is possible to obtain this curve from the theoretical model, as RD is the "ring utilisation factor" and an approximation for Tsat can be obtained from equations (2.6, 2.7, 2.8):

$$Tsat \approx \frac{Q - 1}{\frac{1}{D_i} - \frac{1}{MST}} \quad \ldots\ldots \quad (3.13)$$

3.4 PERFORMANCE VARIABLES

The performance analysis of token passing rings can be achieved by a study of the following three ring **categories**:

(a) An N node ring with only one node active.

(b) An N node ring with all nodes active; where each node has an identical workload and operational characteristics.

(c) An N node ring with all nodes active and N-1 nodes inactive; where each active node has a different workload and operational characteristics.

In each of the three ring cateogories there are a number of variable parameters.:

(i) The number of nodes in the ring: N

(ii) The average message length transmitted by a node (s): S

(iii) The length of the output queue for a node (s): Q

(iv) The snapshot time (where appropriate): Ts.

The ring performance experiments will consist of varying the value of one of the above parameters, whilst maintaining the others at a constant value.

3.4.1 N node ring - one node active

This is conceptually the simplest ring structure. The mean scan time for the token is:

$$MST = S_i + N \qquad \ldots\ldots (3.14)$$

The ring utilisation factor is given by:

$$R = \frac{S_i + N}{D_i} \qquad \ldots\ldots (3.15)$$

Note that $R < 1$ for a stable ring with output queue length, Q.

The parameters of interest are N, S, Q and Ts (in the case of $Q \to \infty$).

3.4.2 N node ring - all nodes active

In this case all nodes are identical, hence the ring utilisation factor is given by

$$R = \frac{N}{D} (S + 1) \qquad \ldots\ldots (3.16)$$

The mean scan time may be computed by substituting D = MST at R = 1, hence:

$$MST = N (S + 1) \qquad \ldots\ldots (3.17)$$

The parameters of interest are S, W, Ts. The value of N is fixed.

3.4.3 *N node ring - some nodes active*

The performance of this type of ring configuration may be assessed by concentrating on one active node only. However, the first two ring **categories** provide a more accurate representation of the performance of a ring under worst case conditions. Therefore, the performance study has concentrated on an analysis of the first two ring categories.

3.5 PERFORMANCE MEASUREMENTS

Performance measurements have been performed on an N node ring with one node active and all nodes active. In all the measurements each node transmits as messages at a constant sending rate.

3.5.1 N node ring - one node active

The effects of varying the size of the ring (N), the message length of the active node (S) and the output queue length (Q) have been studied.

(a) The effect of ring size: N

The first experiment indicates the effect of varying N, whilst maintaining a fixed message length and a fixed (low) queue length. Figure 3.1 indicates that as N increases so the stability point of the ring decreases; that is, the throughput carried for the ring decreases due to the overhead of relaying the token. When the ring saturates the throughput carried decreases due to the limited output queue length. For the purposes of the simulation an active node becomes inactive when its output queue overflows. The ring saturation times for demanded throughputs are given in Figure 3.2. The curves indicate the length of time a ring can achieve an "overload" demand rate before saturation occurs.

By increasing the size of the output queue length, such that the ring does not saturate the curves of Figures 3.3 and 3.4 are obtained, using method 2 and method 3 to determine the throughput carried by the ring, respectively. The snapshot time for each of the experiments was 8 ms (Ts). There appears to be little difference in the results produced by the two methods of measuring the throughput carried by the ring, with the exception that method 3 produces a more realistic result due to an averaging effect. The curves indicate that the throughput saturates and remains constant for $R \geq 1$, which is as expected from the theory.

The effect of total message delay in the ring is indicated in Figure 3.5. The curves illustrate that the message delays increase dramatically as the ring operation enters the saturation region, as predicted by the theory (equation 3.11).

(b) The effect of message length: S

A similar set of curves have been produced to assess the effect of varying S, whilst maintaining a fixed

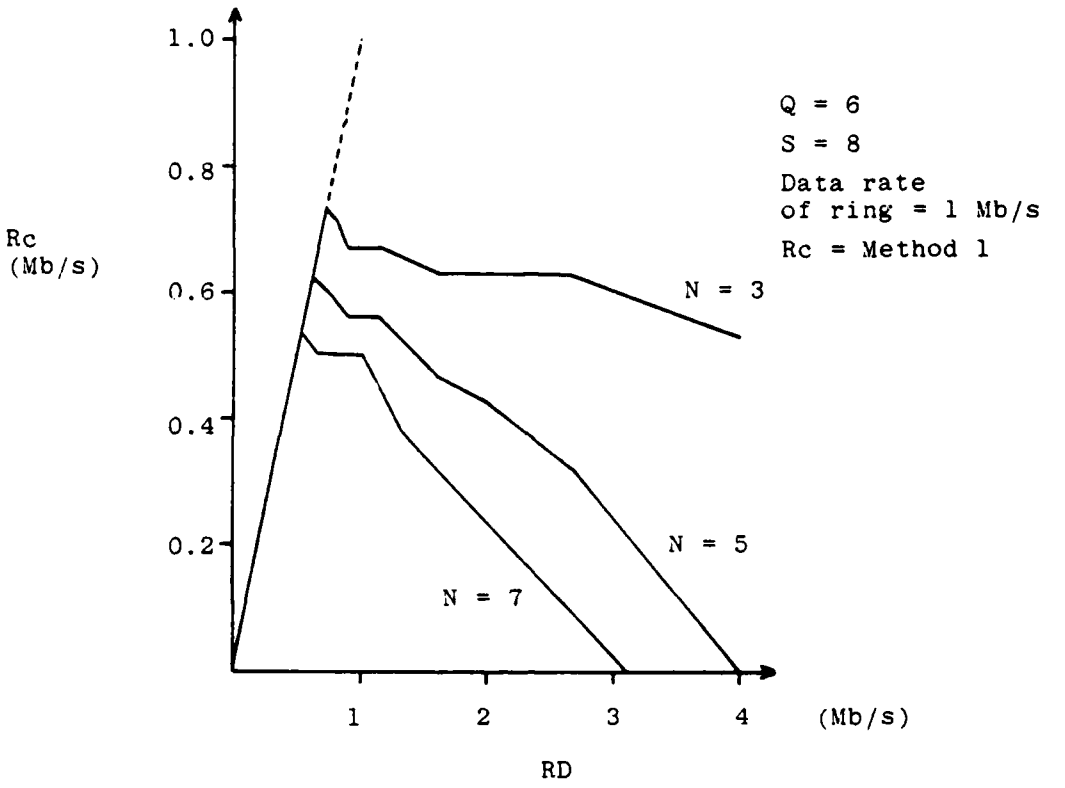


Figure 3.1. Ring Performance 1: Variable N

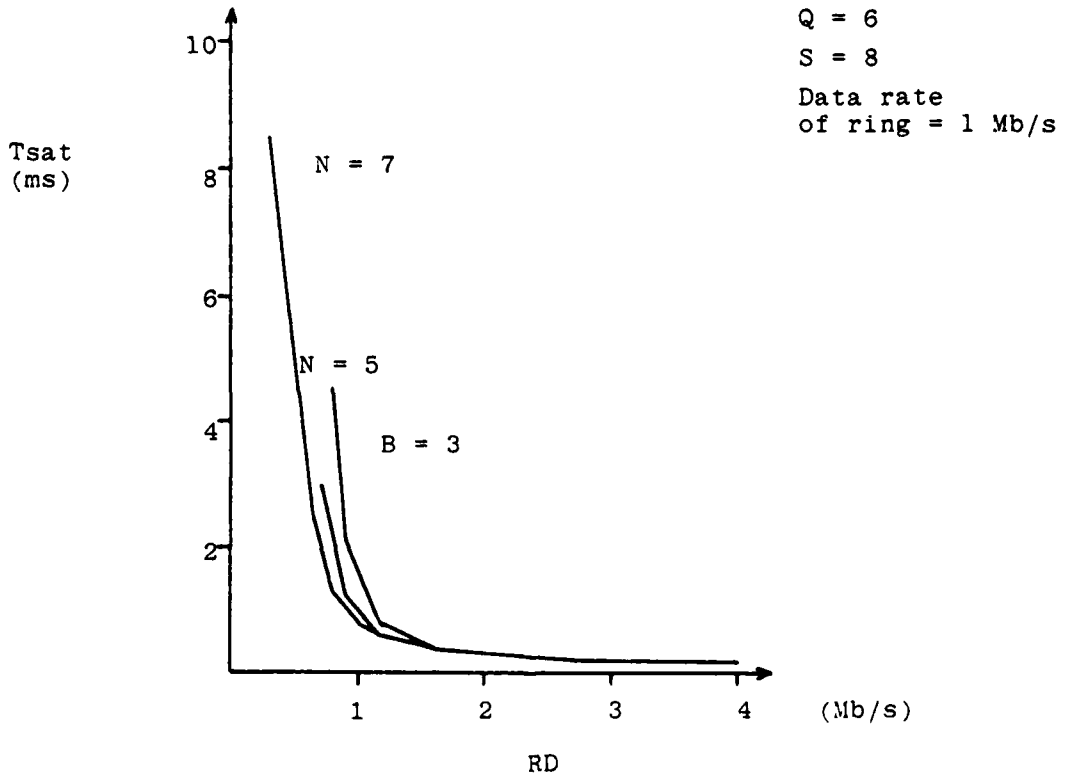


Figure 3.2. Ring Performance 2: Variable N

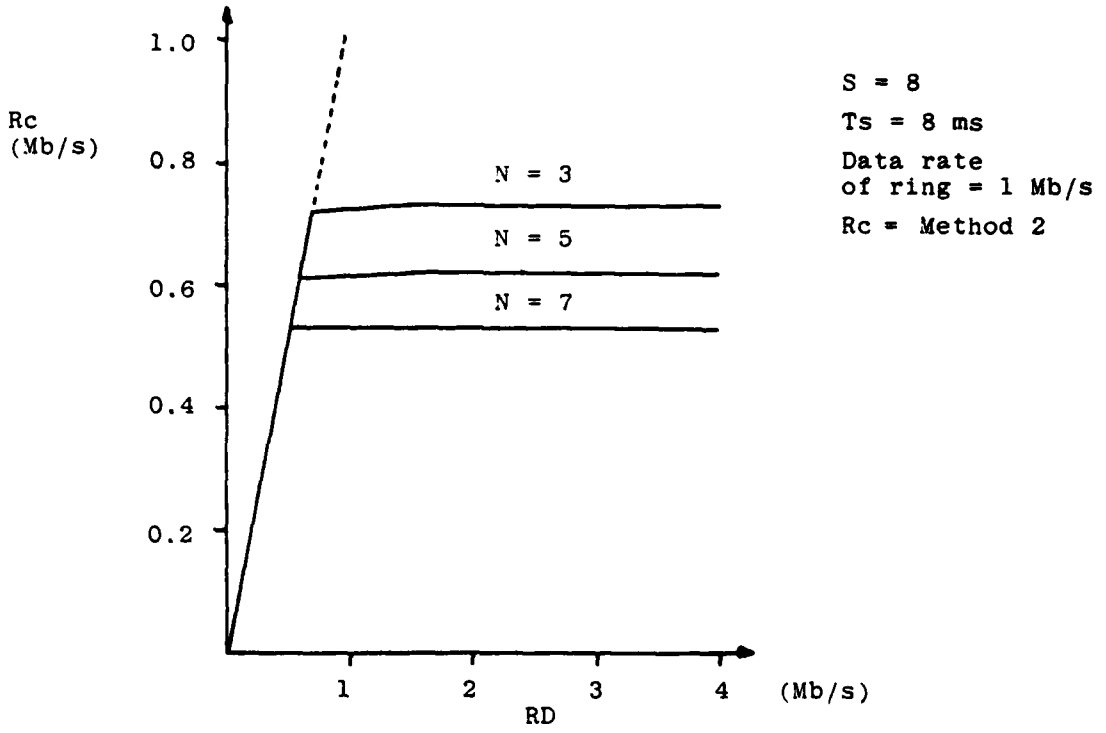


Figure 3.3. Ring Performance 3: Variable N

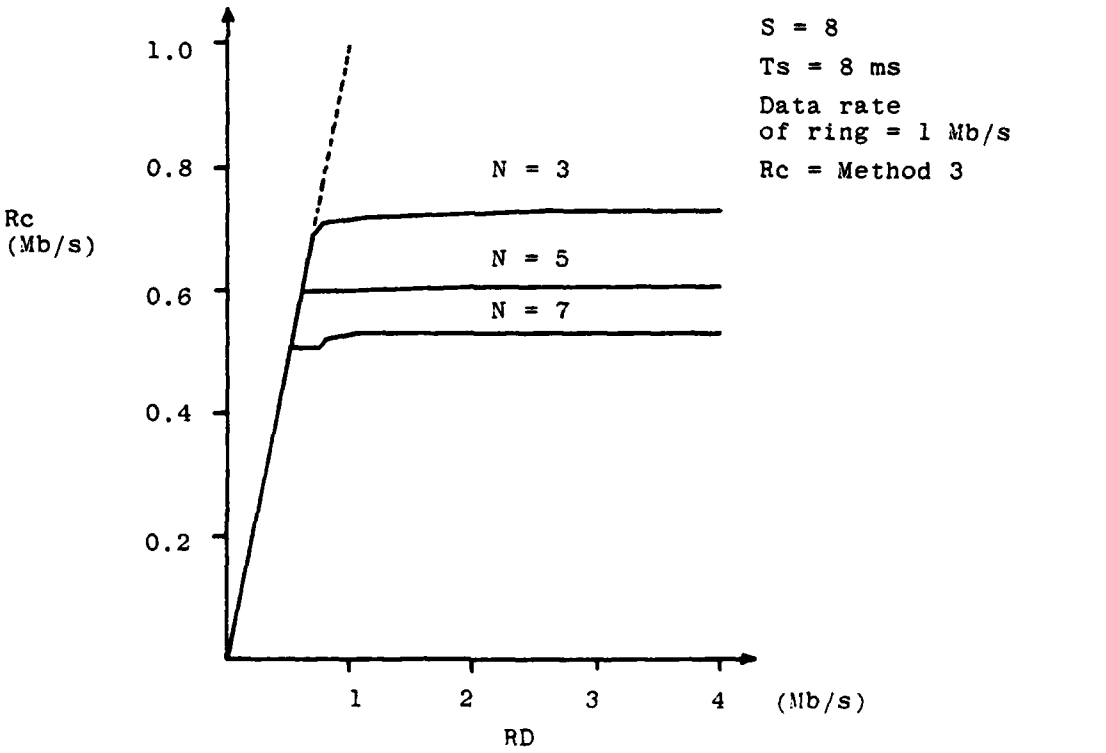


Figure 3.4. Ring Performance 4: Variable N

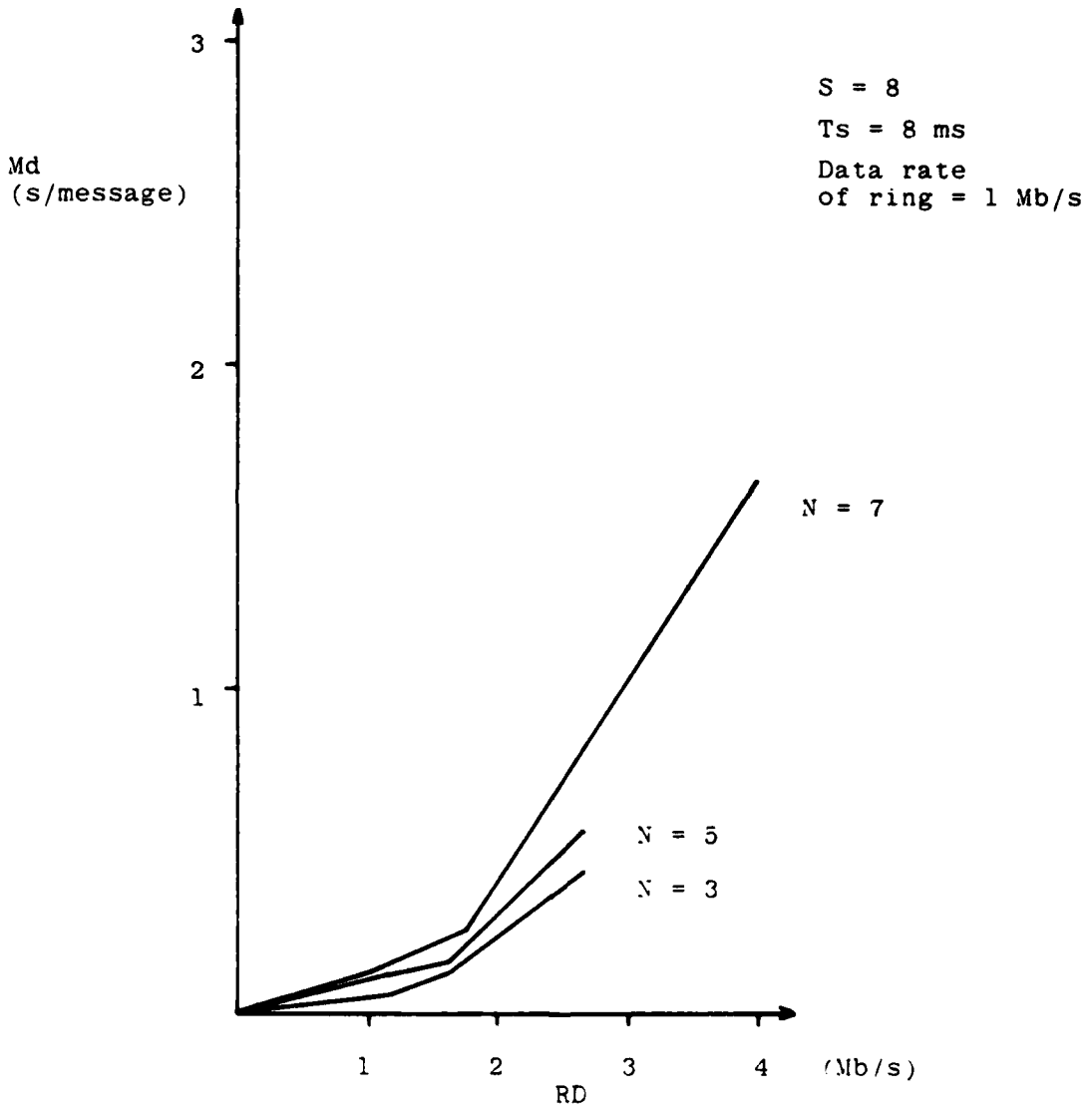


Figure 3.5. Ring Performance 5: Variable N

number of nodes in the ring (N) and a fixed (low) queue length. The corresponding sets of curves are given in Figures 3.6-3.9. It can be seen that as the message length increases, the stability point of the ring increases; that is, the throughput carried by the ring increases. The stability point of the ring is, theoretically, at:

$$RD = \frac{S}{S + N}$$

The performance curves indicate agreement with the theory.

(c) The effect of output queue length: Q

The effect of varying the queue length whilst maintaining a fixed number of nodes in the ring (N) and a fixed message size (S) is indicated in Figures 3.10 and 3.11. The variation in queue length has no effect on the stability point of the ring - all configurations will saturate at the same demanded throughput. However, the ring saturation times are dependent on the queue length, as shown in Figure 3.11.

3.5.2 N node ring - all nodes active

In this study the value of N is fixed, at N = 5. The effects of varying the message lengths (S) and the output queue length (Q) have been studied. Note that the characteristics of all the nodes are identical.

(a) Effect of message length: S

This set of experiments assesses the effect of varying the message length (S) for five active nodes with a fixed queue length (Q). The set of performance curves are given in Figures 3.12-3.15. The curves indicate that as S increases so the ring stability point increases; that is, the throughput carried by the ring increases. The stability point of the ring is, theoretically, at:

$$RD = \frac{NS}{MST} = \frac{S}{S + 1} \qquad \text{as } MST = (S + 1) \text{ at } R = 1$$

(b) Effect of output queue length: Q

The effect of varying the queue length at each node (by the same amount) for five active nodes with a fixed message length (S) is indicated in Figures 3.16 and 3.17. Again the variation in queue length has no effect on the stability point of the ring - all configurations will saturate at the same demanded throughput. However, the ring survivability time in the saturation region depends on the queue length as indicated in Figure 3.17.

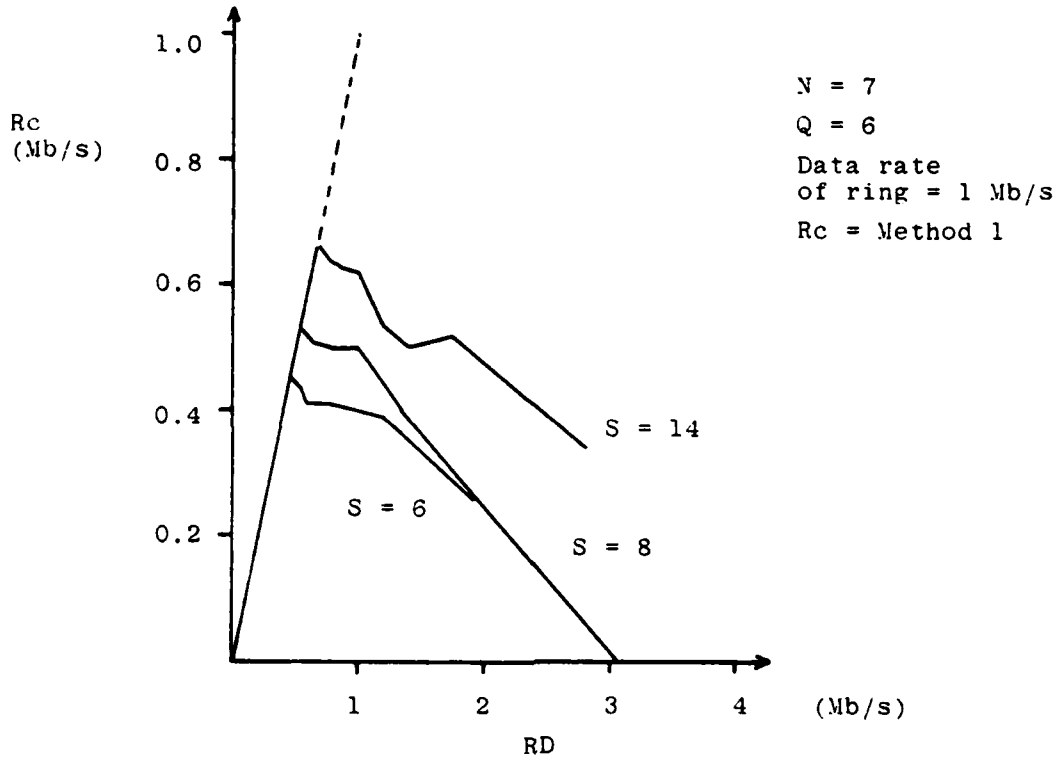


Figure 3.6. Ring Performance 1: Variable S

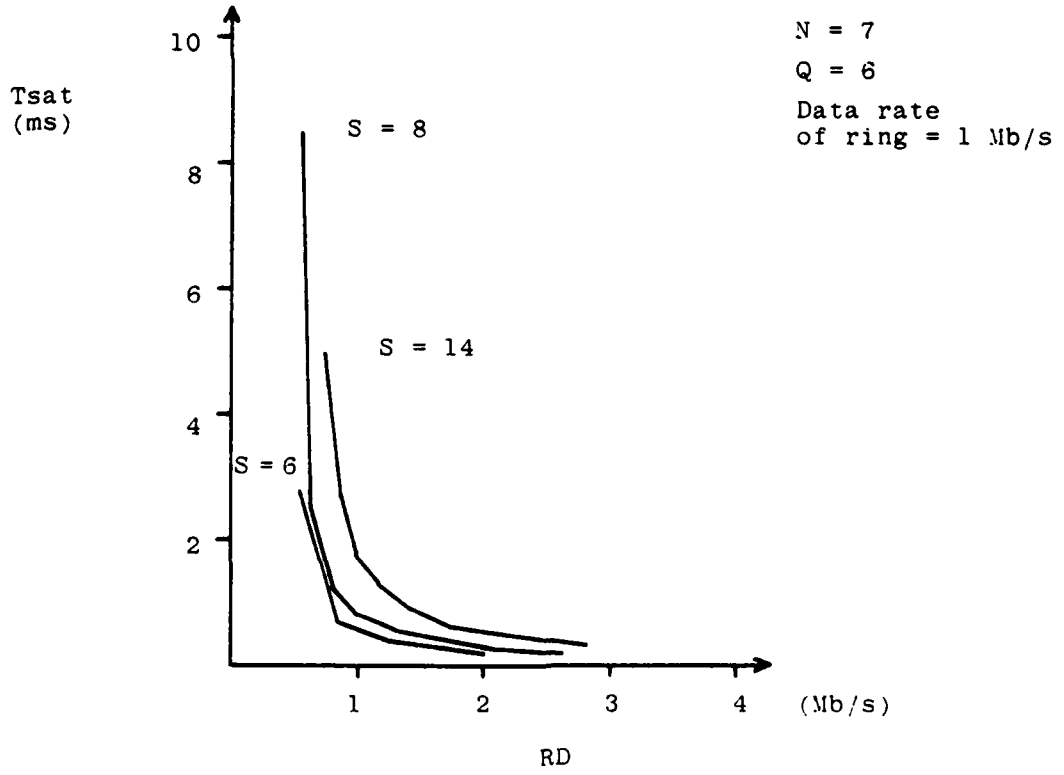


Figure 3.7. Ring Performance 2: Variable S

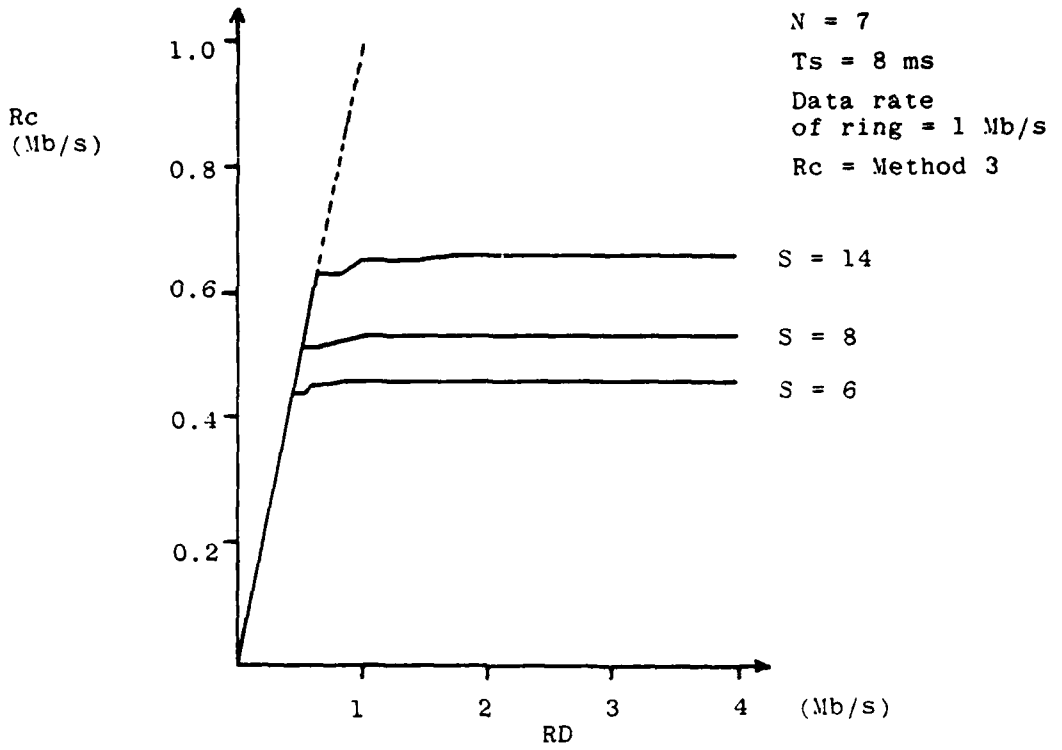


Figure 3.8. Ring Performance 3: Variable S

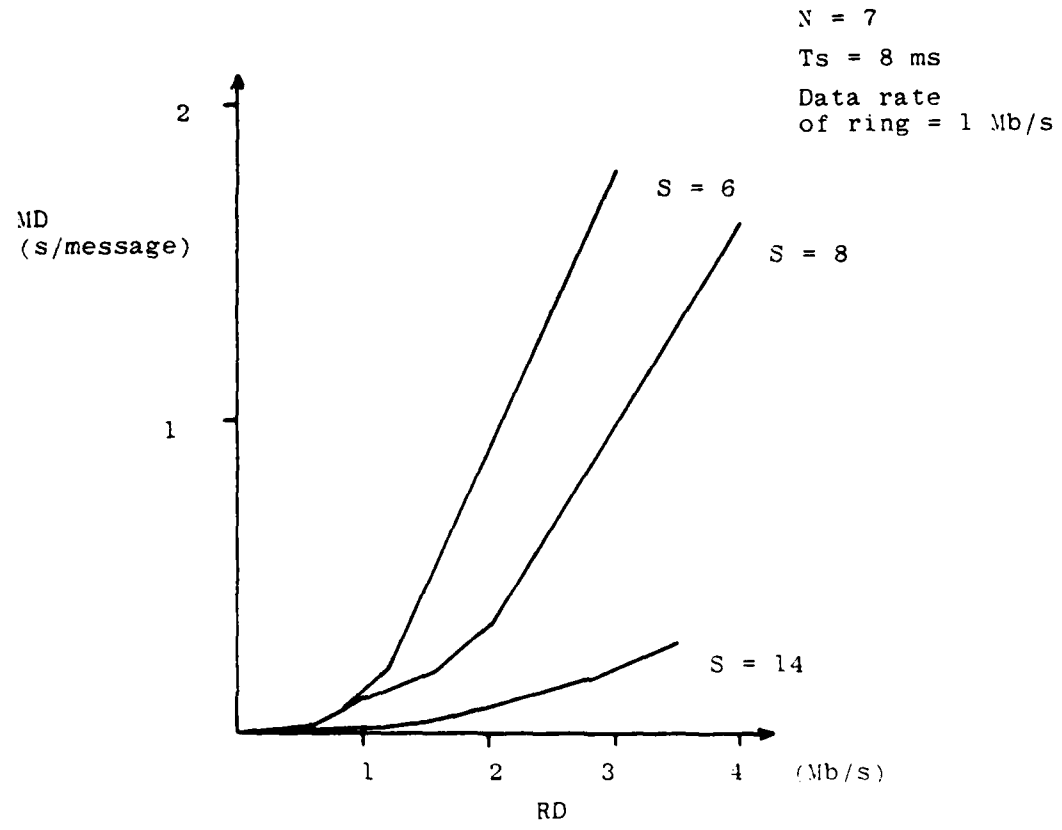


Figure 3.9. Ring Performance 4: Variable S

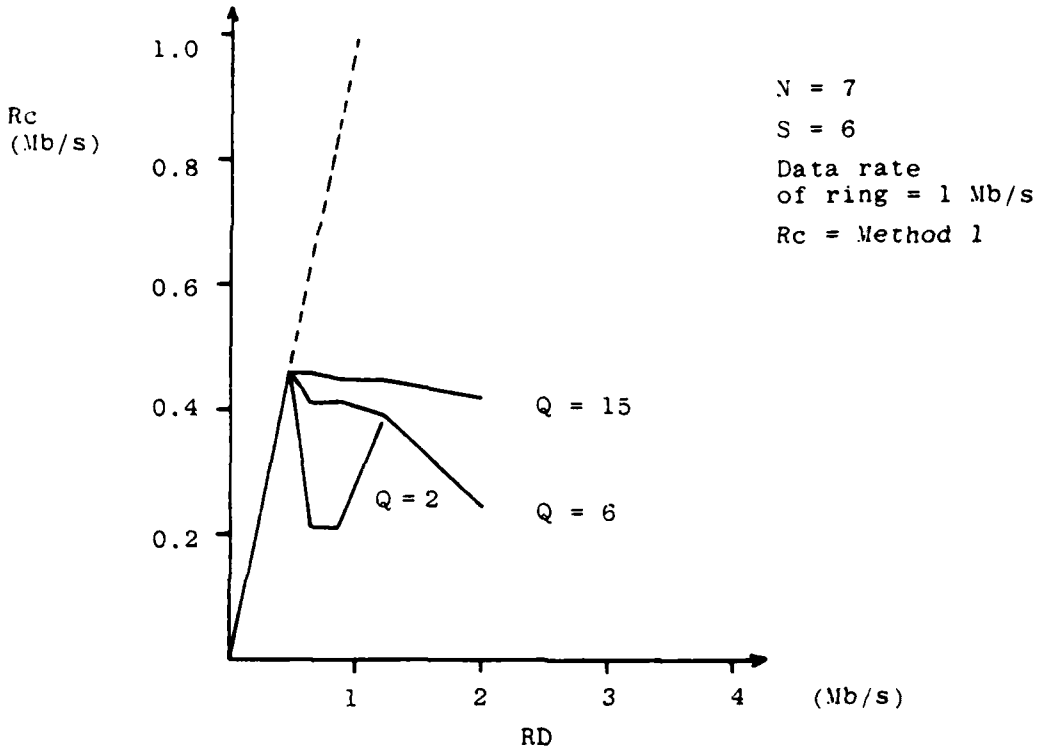


Figure 3.10. Ring Performance 1: Variable Q

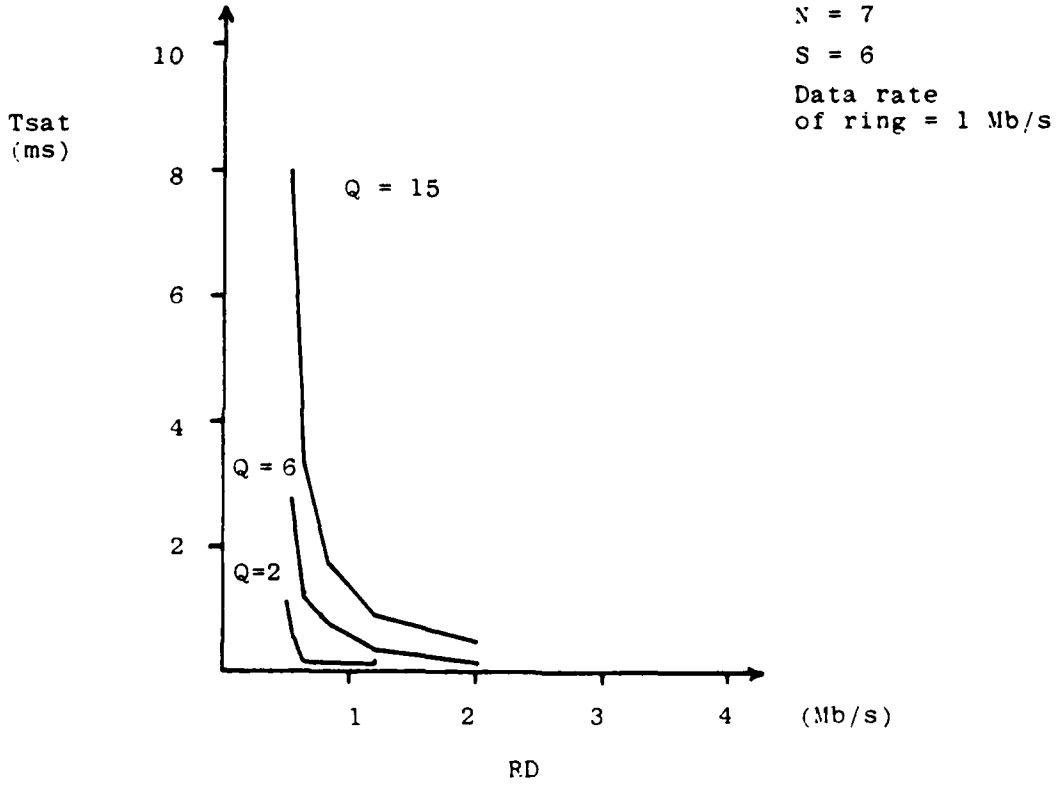


Figure 3.11. Ring Performance 2: Variable Q

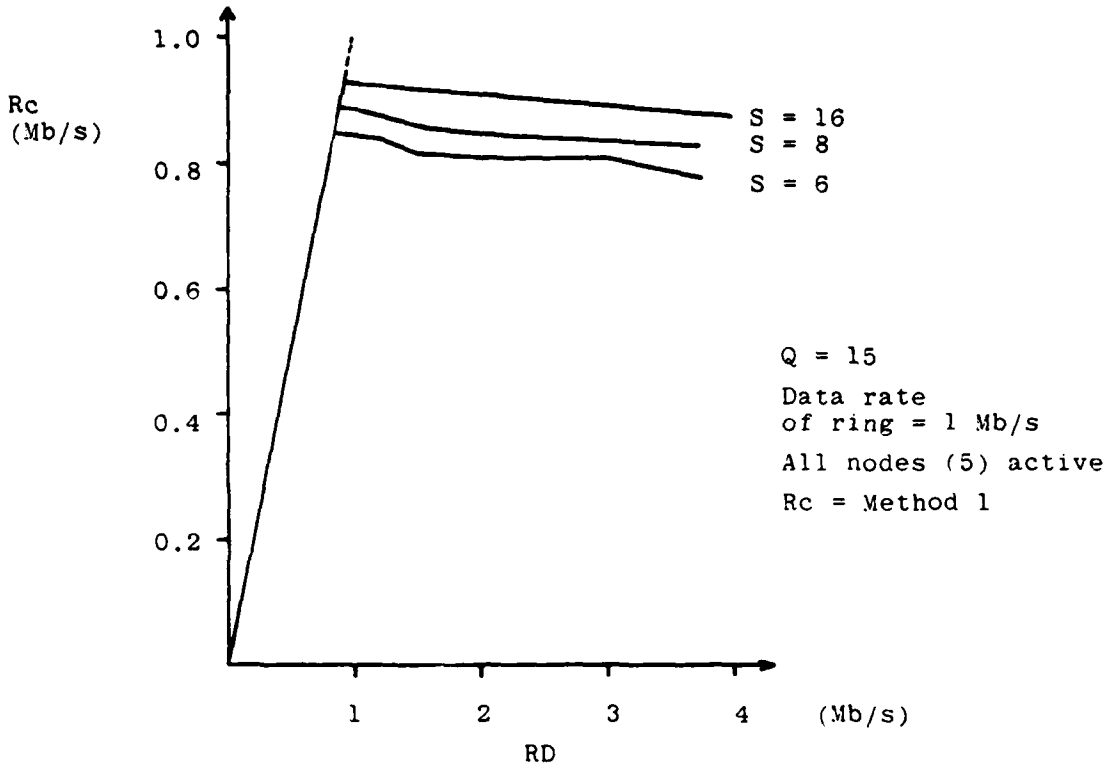


Figure 3.12. Ring Performance 1: Variable S

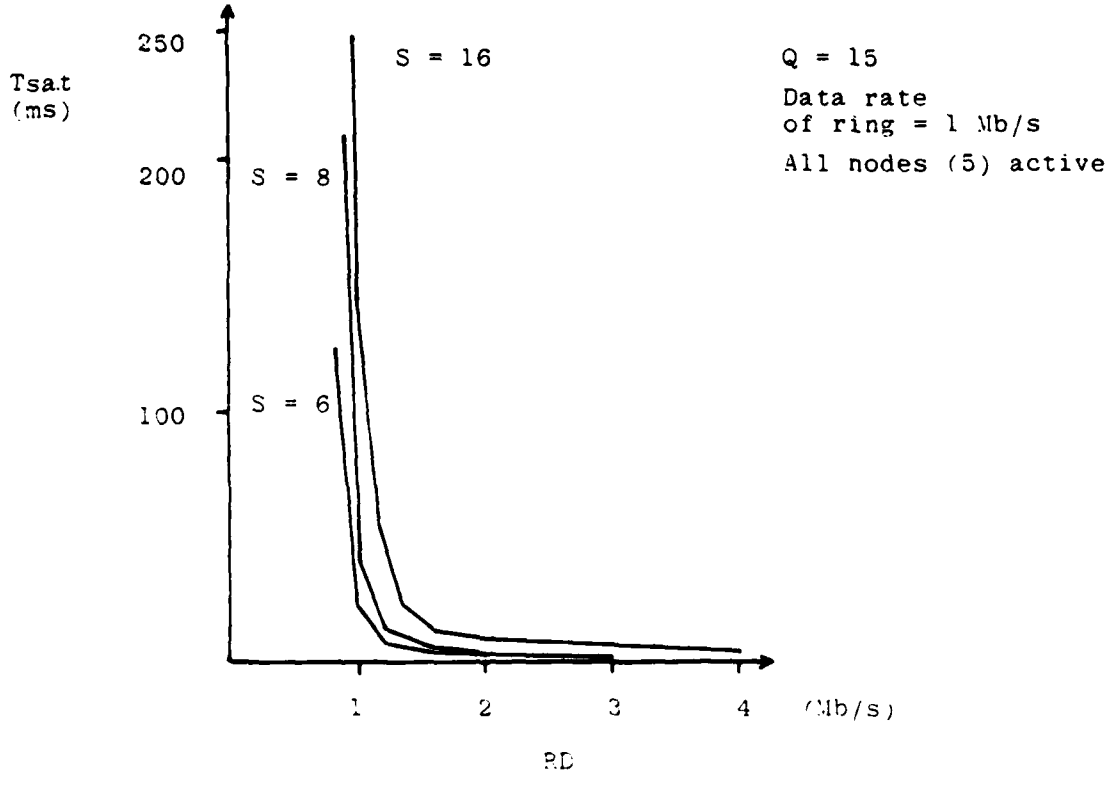


Figure 3.13. Ring Performance 2: Variable S

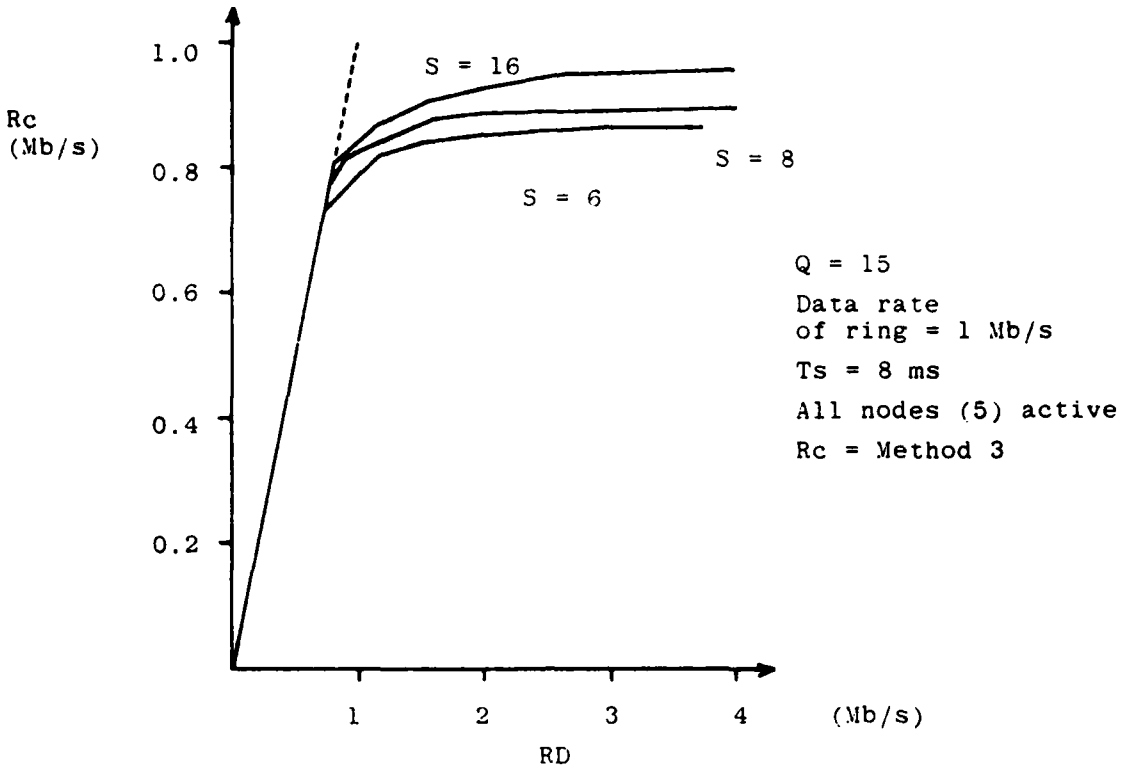


Figure 3.14: Ring Performance 3: Variable S

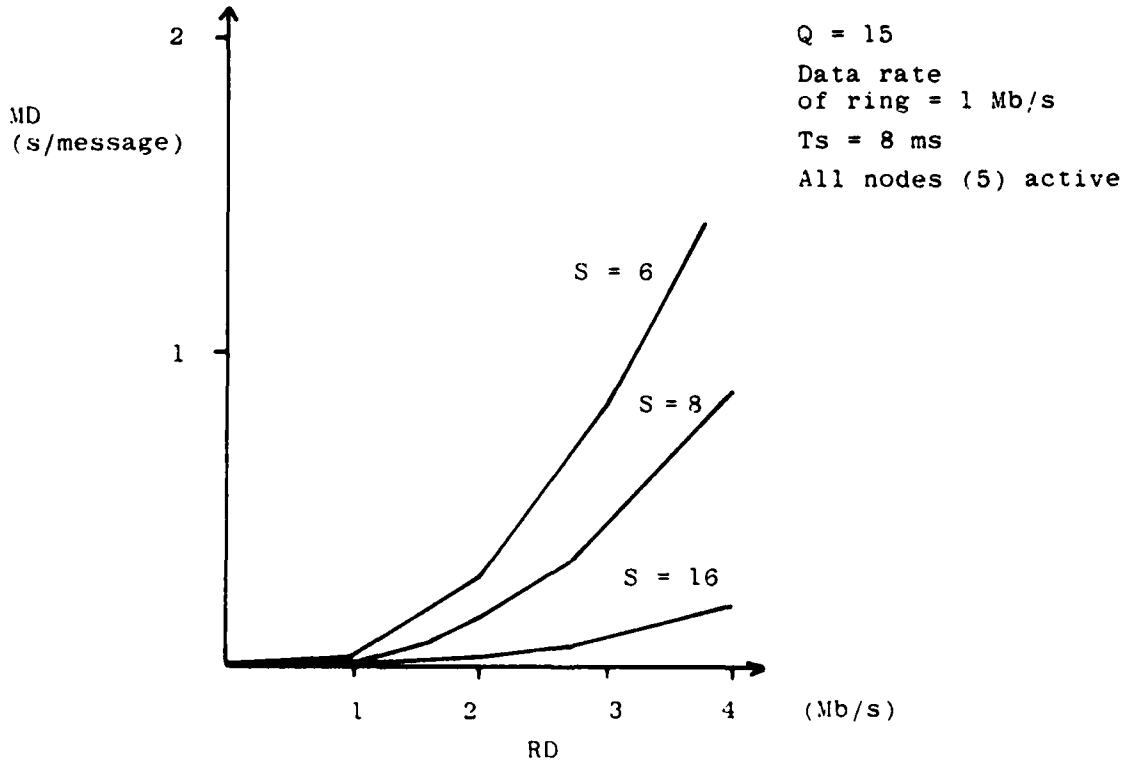


Figure 3.15: Ring Performance 4: Variable S

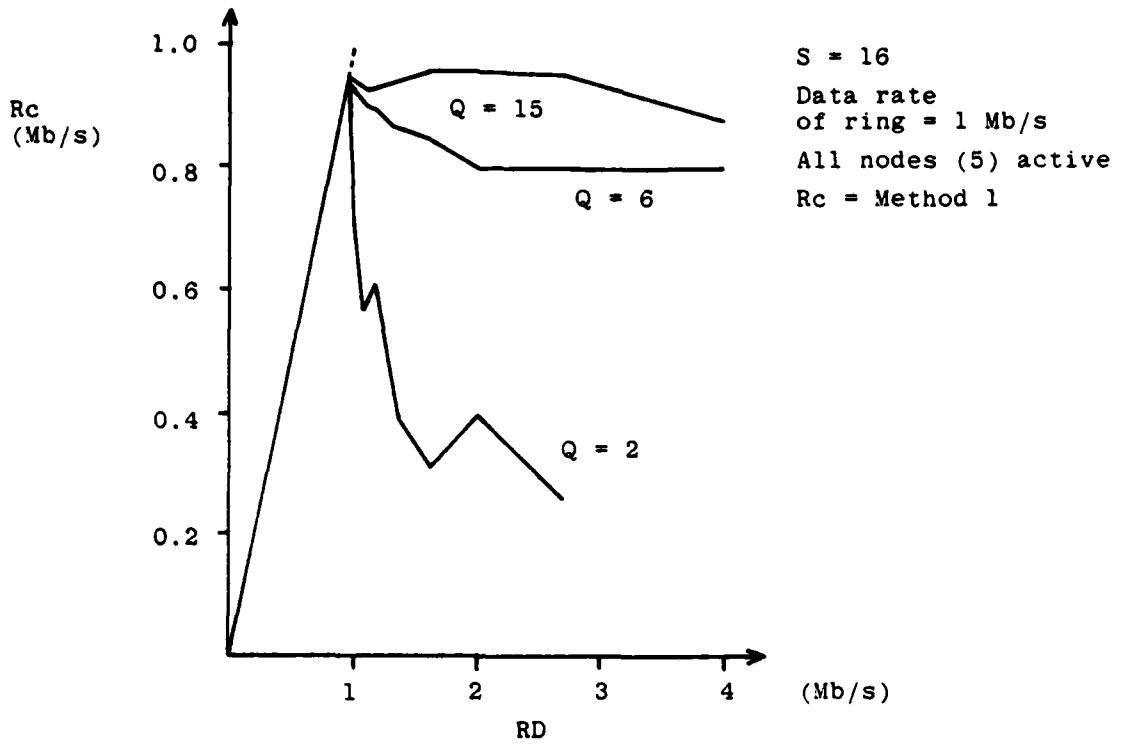


Figure 3.16. Ring Performance 1: Variable Q
Fixed S

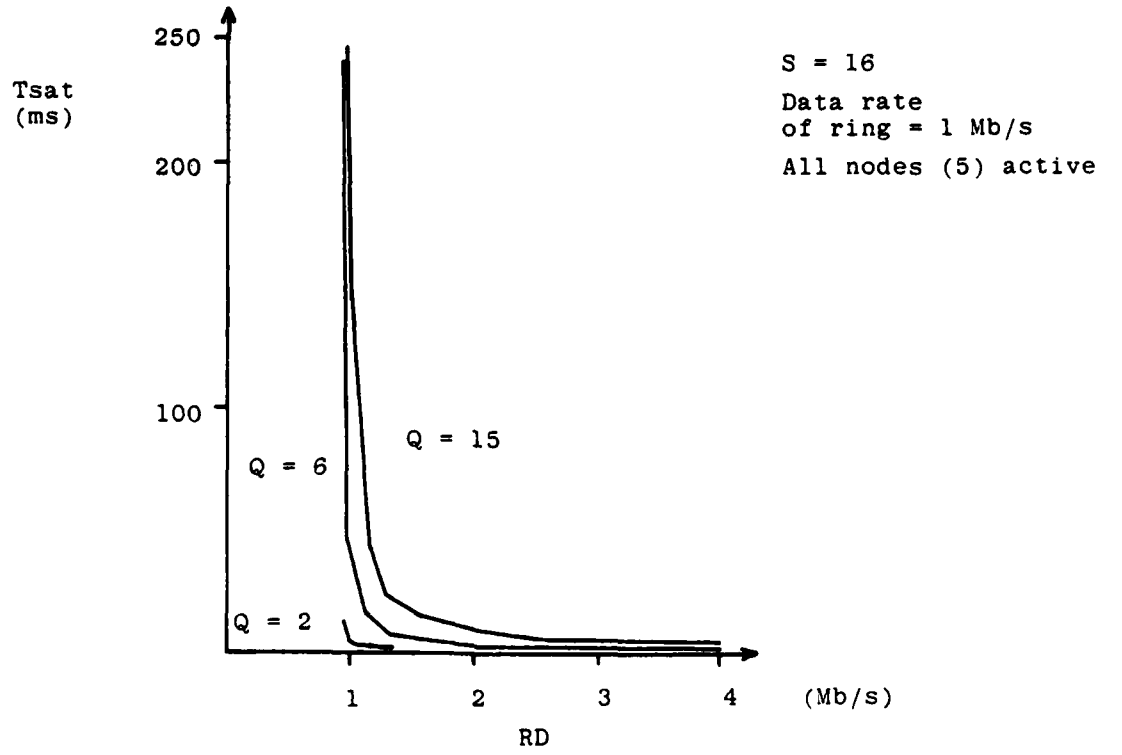


Figure 3.17. Ring Performance 2: Variable Q
Fixed S

3.6 CONCLUSIONS

A set of metrics have been defined for analysing the performance of a ring interconnection structure. Also, a set of standard performance curves have been defined for assessing the characteristics of ring structures under varying workload conditions. The dependent ring variables have been identified and a set of experiments have been performed to characterise the features of a token passing ring structure. The results indicate that the token passing ring is relatively insensitive to variations in workload, possesses short message delays under a light load and controlled message delays under a heavy load.

The experiments have been conducted with each active node having a constant message sending rate. In order to investigate more realistic workloads, and to overcome any ring synchronisation effects, a similar set of experiments will be performed with each active node having a Poisson distributed message sending rate with the same average mean as in the constant message sending rate mode. The theoretical model is also applicable to this type of ring operation.

4. IMPLEMENTATION OF CHILL SIGNALS COMMUNICATION PRIMITIVES ON A DISTRIBUTED SYSTEM

4.1 INTRODUCTION

Reductions in the size and cost of computing power in recent years have encouraged the use of distributed processing for the control of real time systems. This approach offers the advantages of fault tolerance, since processing may be transferred from one element to another, and extensibility, as processing power may be added incrementally. A likely application area for such systems is in telephone exchanges. The typical architecture of a distributed system for real time control is shown in Figure 4.1. A set of processing elements (processor and memory) form the "nodes" and are interconnected by means of some medium often using a ring or bus topology.

The software for distributed sysetms usually takes the form of a number of concurrently executing processes or tasks. These may be allocated statically or dynamically to nodes and there may be one or more processes executing on each node. It is likely that such systems will be programmed in a high level language having in-built primitives for handling the creation of processes and communication between them. One such language is CHILL, designed by the CCITT for programming of telephone switching systems. CHILL provides four distinct methods of inter-process communication, one of which, called "signals", is particularly suitable for distributed systems as it allows data to be sent directly between processes without using explicitly shared buffers.

This section details the results of a study of the implementation considerations for "signals", carried out at UMIST. The principal objective was to examine the problems of implementing "signals" on various distributed architectures. Of particular interest were the issues of partitioning of the system data structures around the nodes, the algorithms used in implementing the primitives and the demands made on the underlying interconnection structures.

The investigation was carried out by "modelling" of the CHILL signals primitives on a multi-microprocessor, CYBA-M. This approach gave the opportunity of studying both an implementation for the CYBA-M architecture itself, which is based on a shared memory, and also the modelling of other architectures.

4.2 THE CHILL SIGNALS PRIMITIVES

CHILL signals are an interprocess communication system of the message passing class. Two primitives are available called "SEND" and "RECEIVE CASE". The SEND primitive is of the simple "send and forget" type - i.e. a signal (the CHILL name for message) is despatched and the sending process then proceeds regardless. The signal remains in the system until it is received by some process instance executing a RECEIVE CASE primitive. A signal has a name which is used for selection of signals in RECEIVE CASE, and may also have an associated data list. The destination process instance may indicate explicitly ("specific destination" node) or, if unspecified, the signal is available for reception by any one process instance executing a suitable RECEIVE CASE ("open destination" node). An optional priority may also be assigned to a signal in SEND. All

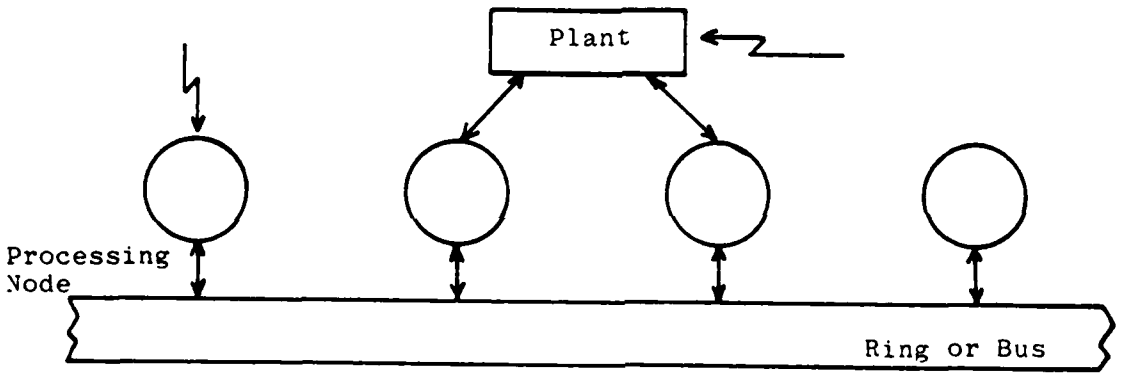


Figure 4.1. Typical Architecture of a Distributed System

signals must be declared like variables before they are used. The declaration may also place a restriction on the possible destination process of a given signalname.

The RECEIVE CASE primitive specifies a set of signalnames that the executing process is interested in receiving. With each set of these may be associated a location for any received data and an action to be performed if that particular signalname is received. An optional "ELSE" clause allows a default action to be carried out in the event that none of the signalnames quoted are present. Otherwise, the instance becomes "delayed" until one of the specified signals arrives. If more than one of the listed signals is available, the one with the highest priority is taken. For further details, the reader is referred to the CHILL specification (15) (16).

4.3 IMPLEMENTATION REQUIREMENTS

An examination of the CHILL specification reveals certain special implementation requirements which must be set by the model. Despite avoidance of explicitly shared buffers in "signals", any implementation of CHILL signals in practice requires some behind-the-scenes buffering of signals and the maintenance of other special information to operate as described in the specification. Signals produced by SEND but not consumed by RECEIVE CASE must be retained somewhere by the system until they are received. The requirements for "open" and "specific" destination signals in this respect are different, as the latter must remain in a common pool for reception by any instance. When a SEND takes place, the status of the destination process - delayed or not - needs to be known, in order that the signal may be delivered immediately or placed in a buffer as appropriate.

From the above considerations, three dynamic data structures were proposed:

(1) Assigned Signal Table; contains signals sent to a specific destination instance, but not yet received. For each one, the destination, signalname, priority, data and instance number of the sender must be stored.

(2) Pending Signal Table; as above, but contains signals sent to open destination.

(3) Delayed Instances Table; contains entries for each process number which is delayed as a result of an unsuccessful RECEIVE CASE. For each one, a list of the signalnames waited for is stored.

In addition to these, two further data structures are needed which, in a simple implementation involving no process creation or destruction, could be static:

(1) Process Name Table; shows the mapping of instance numbers to process names.

(2) Signal Definition Table; for each signalname, lists the permitted process names of its destination if restricted.

Without reference to any specific implementation, the algorithms necessary to implement the CHILL primitives can be expressed in terms of operations on the above data structures. Figure 4.2 is a pseudo-code algorithm showing the main actions necessary for a SEND. Table accesses are indicated by means of the procedures "add" and "remove", which add or remove an instance number or signalname to or from a specified table. The function "search" looks for a signalname in a specified table, returning true if successful. The function "delayed" returns true if the specified instance number is delayed for a given signalname.

First, the parameters of the SEND primitive are checked for possible exception conditions. For example, any conflict between the destination instance and the permitted destination process quoted in the signal definition table is caught. If such a problem is found, the exception handler is called. Otherwise, the algorithm proceeds according to the type of signal - specific or open destination.

If the destination is specific, the Delayed Table is checked to see if the destination instance is delayed. If so, and it wants the signalname in question, the Delayed Table entry is removed (i.e. the status changed to "RUNNING"), the destination instance is reactivated and the signal delivered. If the destination instance was not delayed, the signal is placed in the Assigned Table.

For an open destination signal, the procedure is similar, but this time the delayed table must be searched to find any instance which is delayed for the signal. If a suitable instance is found, the table is updated and the destination instance reactivated as above. If no suitable delayed instance exists, the signal is placed in the Pending Table.

Figure 4.3 illustrates the algorithm for RECEIVE CASE. A receiving process begins by searching the Assigned Table for the signalnames listed in the RECEIVE CASE construct. If a matching entry is found, the signal is removed from the table and the receive terminates. If the search is unsuccessful, the Pending Table is similarly searched. Note that by this ordering, ASSIGNED signals are given priority over PENDING signals. This is logical since the latter can be dealt with by another instance. Within each search, the highest priority suitable signal is taken where a choice exists. If the second search also fails, then the algorithm checks for the presence of an ELSE clause. If present, the action specified after "ELSE" is carried out. With no "ELSE", the flag for the receiving instance in the Delayed Table is set (i.e. the status changed to "DELAYED") and the instance sleeps until awoken by a suitable sender. When this happens, a signal is received and the action specified for the matching signalname is executed.

If all the processes were allowed to access the three tables concurrently, inconsistencies would arise. For example, two senders may pick the same destination instance from the delayed table and both attempt a reactivation operation, or two receivers may both try to take the same signal from the Assigned or Pending tables. Thus any practical implementation must in some way restrict accesses to provide mutual exclusion for key operations on the database.

```
PROCEDURE SENDSIGNAL( Signal );
BEGIN
    IF Signal,destination = SPEICIFIC THEN
        IF processname( Signal,destination ) = EXISTS
                AND (processname( Signal,destination ) =
                    pnamerestriction( Signal,name )
                OR pnamerestriction( Signal,name ) = NONE) THEN
            LOCK Tables;
            IF delayed( Signal,destination for Signal,name ) THEN
                removed( Signal,destination from Dtable );
                UNLOCK Tables;
                reactivate( Signal,destination with Signal,name );
            ELSE
                IF full( Atable ) THEN
                    exception( OVERFLOW );
                ELSE
                    add( Signal to Atable );
                FI
                UNLOCK Tables;
            FI
        ELSE    exception( MODEFAIL );
        FI
    ELSE    (open destination)
        LOCK Tables;
        Found := FALSE:
        FOR Instance := Firstinst to Lastinst WHILE NOT Found DO
            IF delayed( Instance for Signal,name ) AND
                (processname( Instance ) = pnamerestriction( Signal,name
                OR pnamerestriction( Signal,name ) = MORE ) THEN
                    Found := TRUE;
            FI
        OD
        IF Found THEN
            remove( Instance from Dtable );
            UNLOCK Tables;
            reactivate( Instance with signal );
        ELSE
            IF full( Ptable ) THEN exception( OVERFLOW );
            ELSE add( Signal to Ptable );
            FI
            UNLOCK Tables;
        FI
    FI
END
```

Figure 4.2. Algorithm for the SEND Signal Operation

```
PROCEDURE RECEIVECASE( Signalnamelist, Actionlist );
BEGIN
    LOCK Tables;
    Found := FALSE;
    FOR Signal,name IN Signalnamelist WHILE NOT Found DO
        IF search( Atable for Signal,name ) THEN Found := TRUE; FI
    OD
    IF Found THEN
        remove( Signal from Atable );
        UNLOCK Tables;
        call( address in Actionlist for Signal,name );
    ELSE
        Found := FALSE;
        FOR Signal,name IN Signalnamelist WHILE NOT Found DO
            IF search( Ptable for Signal,name ) THEN Found := TRUE; FI
        OD
        IF Found THEN
            remove( Signal from Ptable );
            UNLOCK Tables;
            call( address in Actionlist for Signal,name );
        ELSE
            IF Else IN Actionlist THEN
                UNLOCK Tables;
                call( address in Actionlist for Else );
            ELSE
                add( This instance, Signalnamelist to Dtable );
                UNLOCK Tables;
                Signalname := sleep( until reactivated );
                call( address in Actionlist for Signalname );
            FI
        FI
    FI
END
```

Figure 4.3. Algorithm for the RECEIVE CASE Operation

4.4 MODELLING OF THE CHILL SIGNALS PRIMITIVES

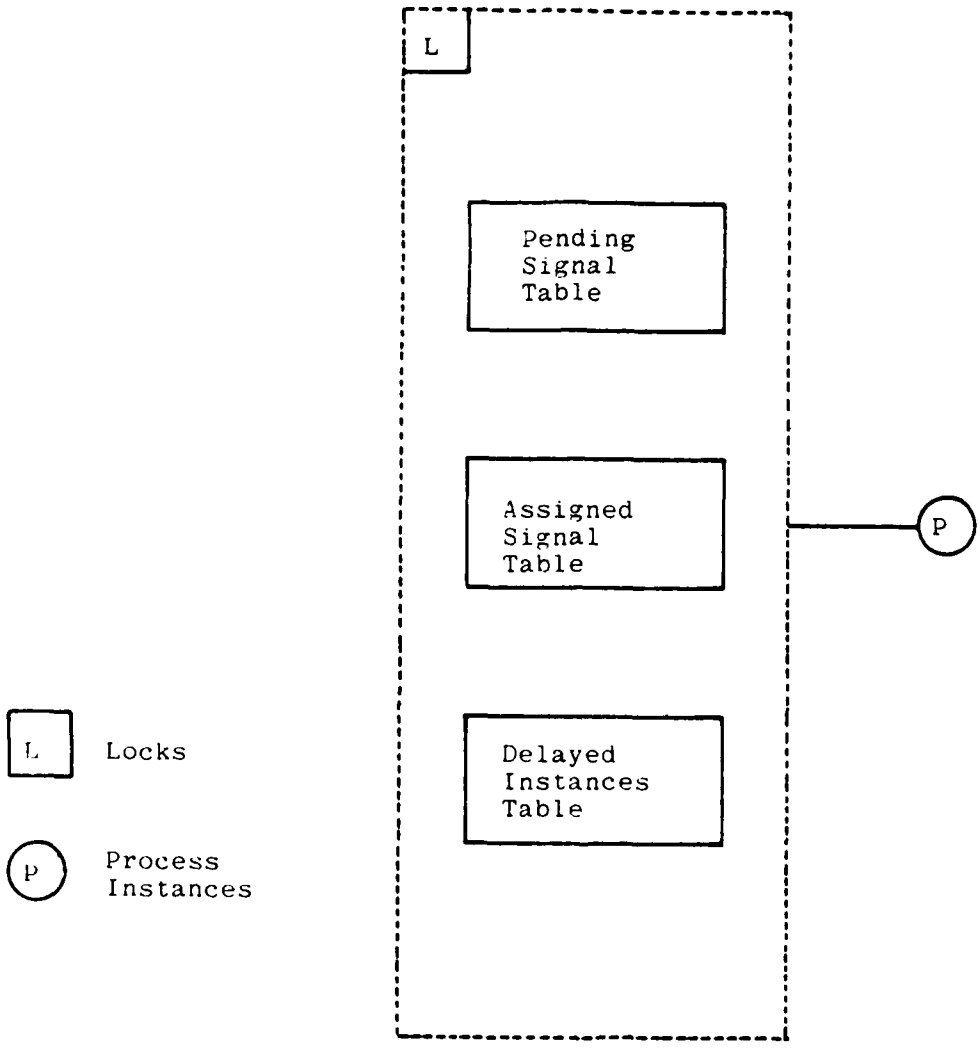
A multi-microprocessor, CYBA-M (17),was used to test various possible implementation strategies for the algorithms discussed above (18). Performance measures were devised and exercised on each of four "models". The first three of these all assume the existence of shared memory - in fact they use the native architecture of the CYBA-M multiprocessor, Direct Shared Memory (DSM). These variants of the DSM implementation differ in the schemes used for mutual exclusion over accesses of the datastructures:

(1) The first implementation used the simplest possible strategy. When any process instance requires access to datastructures, they are locked as a whole using a single binary semaphore. Thus only one process instance can access the tables at any time, as shown in Figure 4.4(a). The algorithms of Figures 4.2 and 4.3 were shown with this locking protocol - the operations LOCK and UNLOCK convey waiting for the semaphore (P) and releasing it (V) respectively. Naturally, this scheme is very restrictive, and leads to processes spending most of their time waiting on locks for access to the datastructures.

(2) For the case of specific destination signals; the Assigned and Delayed tables can be divided by destination instance number, and each section given its own separate lock. Unfortunately, the Pending table cannot be similarly divided, since it contains signals for potential reception by any process instance. In this first variant of the simple model, multiple locks are used for the Assigned and Delayed Tables as shown in Figure 4.4(b). A set of locks (one per instance) covers both of these tables for protection during concurrent receive or send and receive operations. Another lock set covering only the Delayed table provides mutual exclusion between senders checking the status of potential receivers. The Pending table is given a single separate lock.

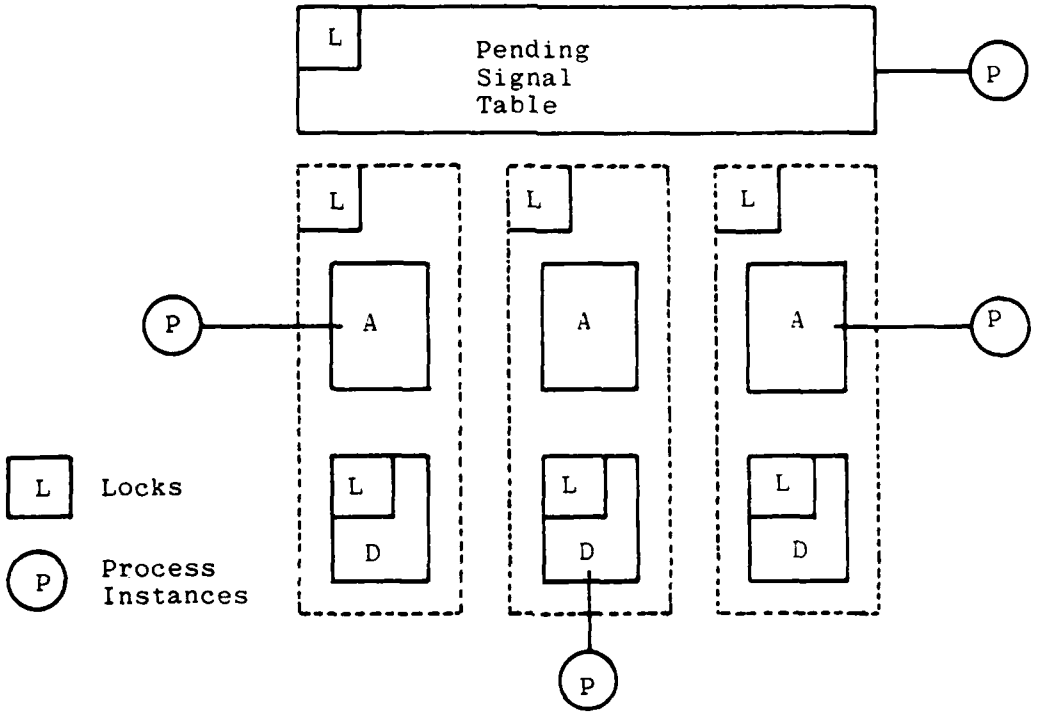
(3) In a system with shared memory, contention for the Pending table can be reduced by allowing multiple readers or one writer to access this datastructure, as shown in Figure 4.4(c). This means that any number of receiving instances may carry out searches of the Pending table at the same time, though a sender or receiver updating an entry must obtain exclusive access. Since table searching is the most time-consuming operation in the SEND and RECEIVE CASE algorithms, this approach significantly increases parallelism. A second implementation variant was produced using this scheme. The locking arrangements for the Assigned and Delayed tables remain as in (2).

The above implementations all rely on shared memory for their operation. In a typical distributed system the nodes will not have any common address space, however, so the consequences of implementing the CHILL primitives without this convenience were investigated. The arrangement of (2) above maps readily onto a non-shared memory system, with each Assigned and Pending table segment placed in the local memories of the nodes. The principal

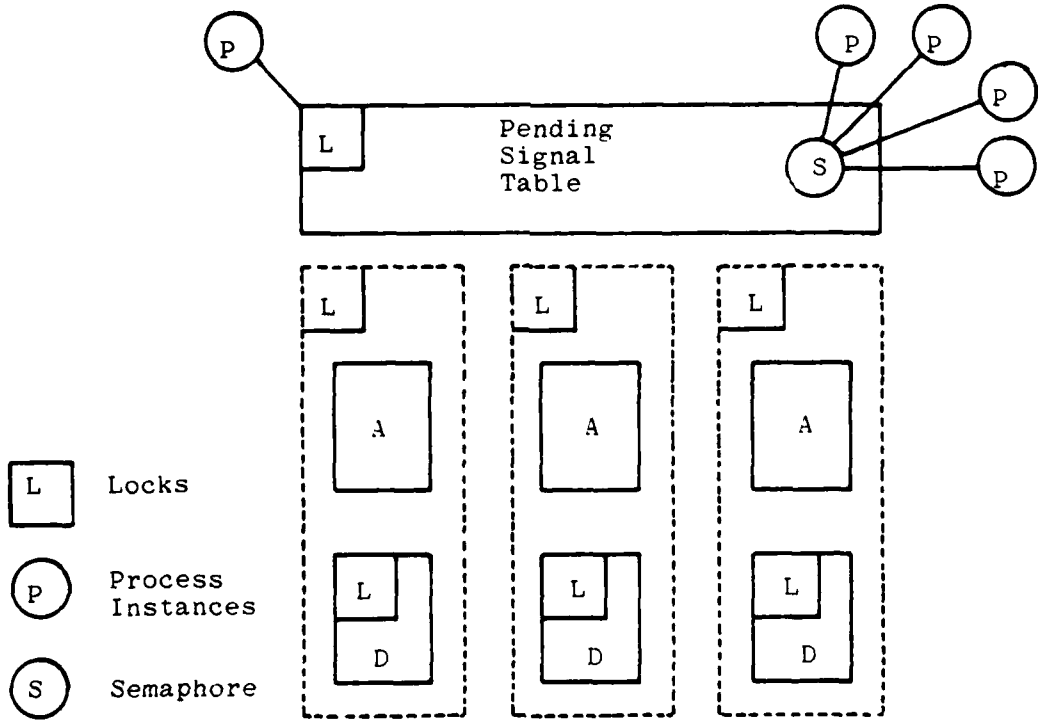


(a)

Figure 4.4. Mutual Exclusion Mechanisms for Access to the Data Structure


L
Pending
Signal
Table
P
L
A
D
Locks
Process
Instances

(b)


P
L
Pending
Signal
Table
S
A
D
Locks
Process
Instances
Semaphore

(c)

trouble area is the open destination signal type. Without shared memory, the method of (3) for Pending table access cannot be used. Instead, this table must be distributed around the nodes. Assuming that a purely symmetrical solution is required (i.e. no special centralised distributed node), there are two main approaches to this:

(a) Replication of the Pending table at all nodes, each open-destination signal sent out being replaced in all of the copies. A scheme for ensuring consistency of the copies would be necessary (as in a distributed database), which is a non-trivial problem.

(b) Each node could retain all open destination signals which it sends out in a local buffer, which would be polled by all other nodes executing a RECEIVE CASE.

Both of the above are rather unattractive and would be difficult to implement efficiently.

A model was produced to test an implementation without shared memory for specific destination signals only. Each node was represented by two CYBA-M processors as shown in Figure 4.5, one acting as a "Host" processor and the other a "communications" processor. The host processor is responsible for the execution of applications programs and RECEIVE CASE operations. When a SEND takes place, it will transfer the parameters of a signal across the network to the communications processor at the destinations node, which buffers up the signals in a local Assigned table for its associated host. Within each node, the two processors are assumed to share address space, and a single lock provides mutual exclusion between them.

4.5 RESULTS

4.5.1 Throughput Characteristics

The four models were compared using a "benchmark" workload program, which simulated the arrival of send-requests at a chosen average rate. The inter send-request delays had a Poisson distribution about the nominal desired value. The mean send-request rate gives a figure for the throughput demanded or offered to the system. The actual rate at which SEND primitives were executed was also measured, yielding the throughput carried. The latter was recorded for a range of values of throughput demanded, resulting in the characteristics shown in Figure 4.6.

Curve A represents the performance of the "single lock" implementation (1), using the six process instances, grouped as three sender-receiver pairs. The behaviour for open and specific destination signals in this case is identical. Note that the implementation fails before reaching saturation - this is due to overflow of the system tables. The contention for the datastructures is so severe that receivers cannot gain access sufficiently quickly to empty them.

Curve B is the throughput characteristic for specific destination signals using the multi-lock model (2). The performance is greatly improved compared with (1). Table overflow no longer occurs and a saturation throughput of 93 signals per second is achieved, which is about six times the maximum obtained using the single lock. Saturation is simply occurring when the average inter-send delay becomes less than the time taken to execute the

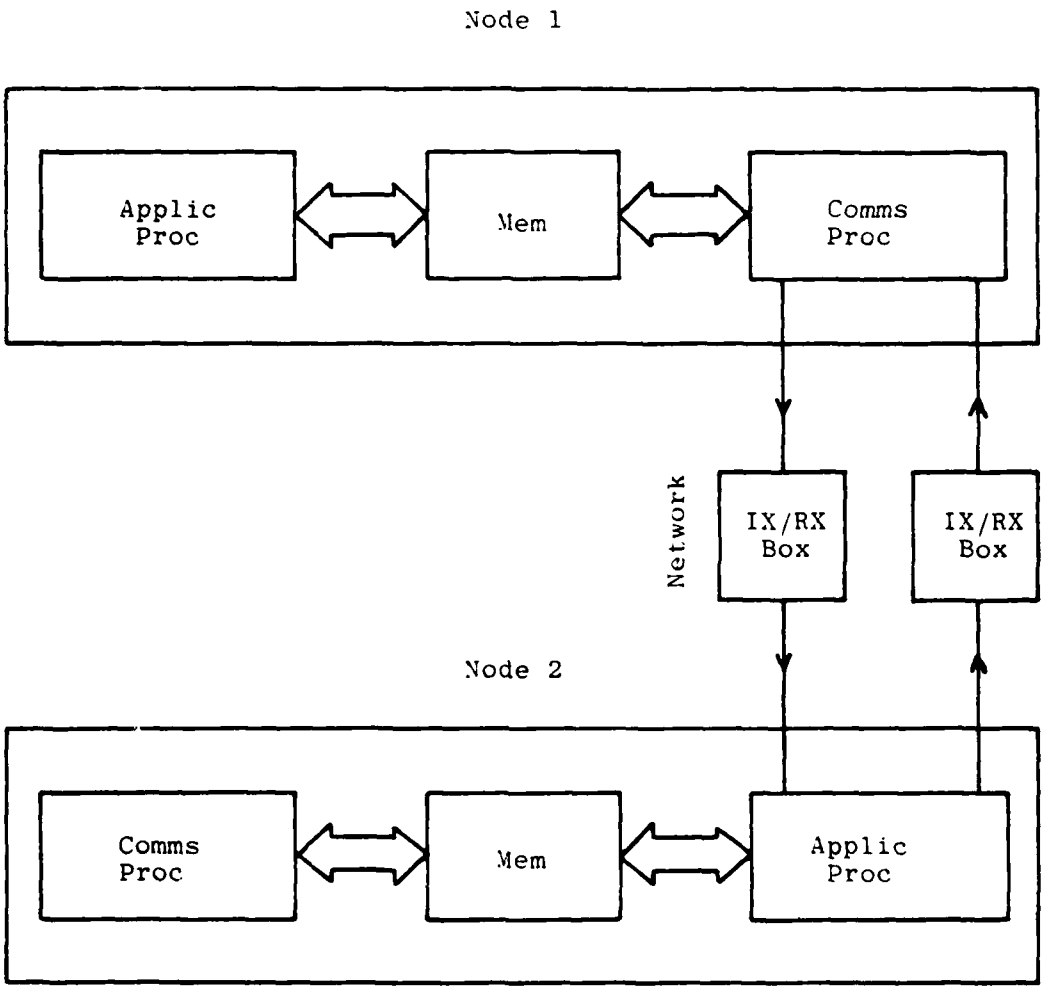


Figure 4.5. Model Architecture

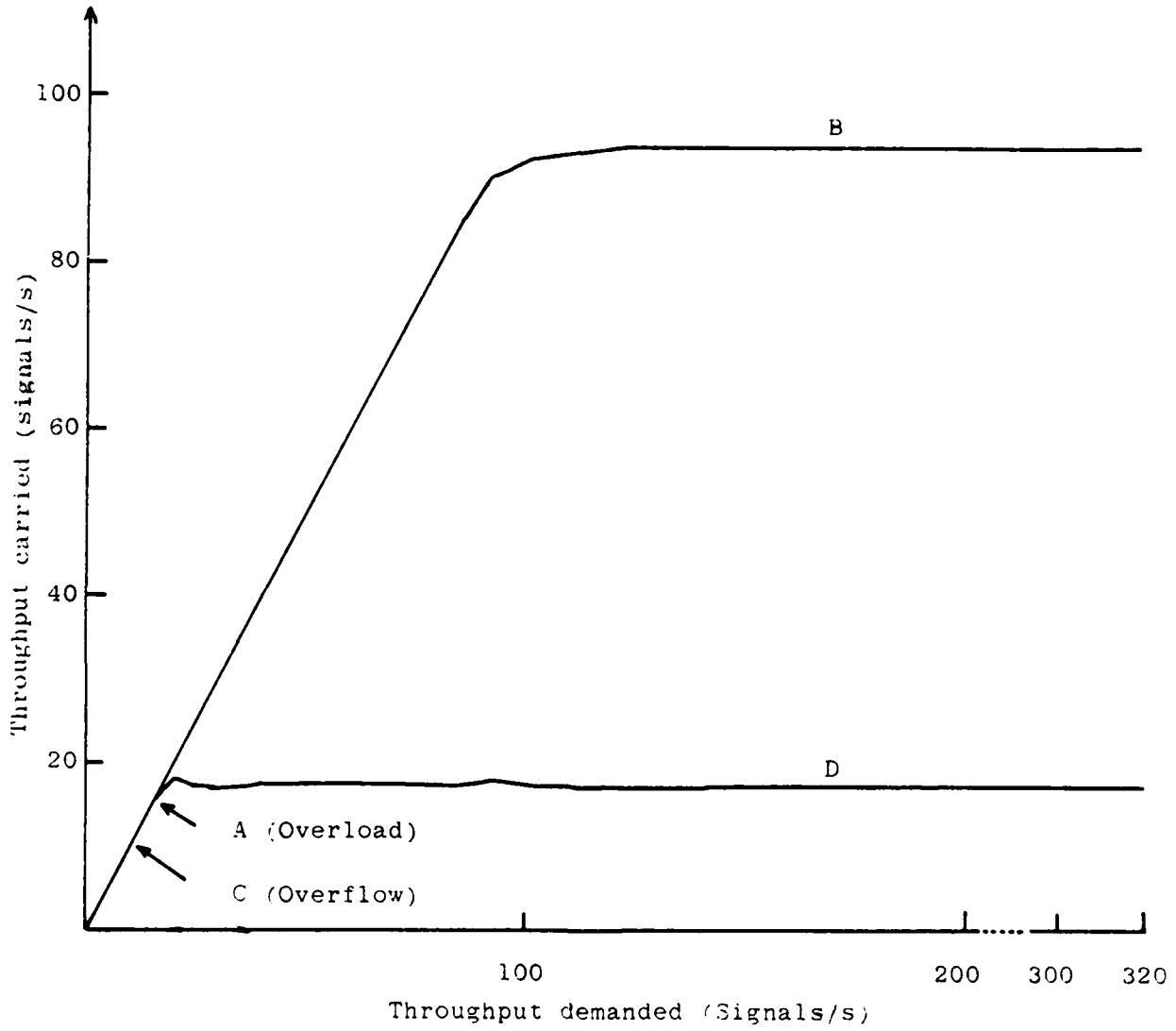


Figure 4.6. Throughput Characteristics of the DSM Models

SEND operation itself - there is no build-up of signals in the tables.

Using open destination signals (curve C), however, model 2 presents no improvement, as expected since the Pending table still has only one lock. In fact, overflow occurs at a slightly lower throughput for (2), probably due to the extra overheads involved in checking the Assigned table where multiple locks now have to be obtained (for no advantage since it will be empty when all signals are open destination).

Curve D shows the performance of the multi-lock multi-reader model, (2) with open destination signals. It can be seen that overflow no longer occurs, though the saturation throughput is still inferior to the specific destination case. (The performance for specific destination signals is the same as that for model (2) - curve B. To summarise, both variants of the original model remove the constraints on performance for specific destination signals, but the saturation throughput attainable with an all-open destination workload is limited.

The non-shared memory model has extra overheads compared with the above three, due to the need to transmit each word of a message over the simulated network. However, the use of two processors per node allows some extra parallelism to be exploited, reducing the processing required on the host processor during a SEND. If the delays in the network are assumed to be negligible, the overall result is a performance very similar to Figure 4.6 Curve B, with a slightly lower saturation throughput of 80 signals per second. In practice, this would of course be degraded by the delays in a real network.

4.5.2 Effect of Signal Priority

The transmission delay suffered by signals in the models was also measured. This is the delay from the time of a send-request to the time the corresponding signal is received. An average figure was recorded over the range of "throughput demanded" for each of the four priority levels implemented in the models. The priority levels are numbered from 0 (lowest) to 3 (highest). A typical result is shown in Figure 4.7. As expected, the transmission delay at any particular throughput level is smaller the higher the priority. Also, the differences between those delays increase with throughput carried. More significantly, however, the delay rises more rapidly for low priority signals than high ones, a pattern which gives cause for concern. It arises because when there is a build-up of signals in a table, the probability of a signal being chosen by a receiver falls rapidly with decreasing priority. It was also noticed, when the models were running certain workloads, that it was possible for some low priority signals to remain in tables indefinitely. New, higher priority signals were arriving between each RECEIVE and these were taken in reference to the low priority ones. If this pattern is maintained, the latter may never be received.

4.6 CONCLUSION

This study allows some comments to be made on the CHILL communications primitives themselves; their usefulness, problem areas and the constraints they impose on system design.

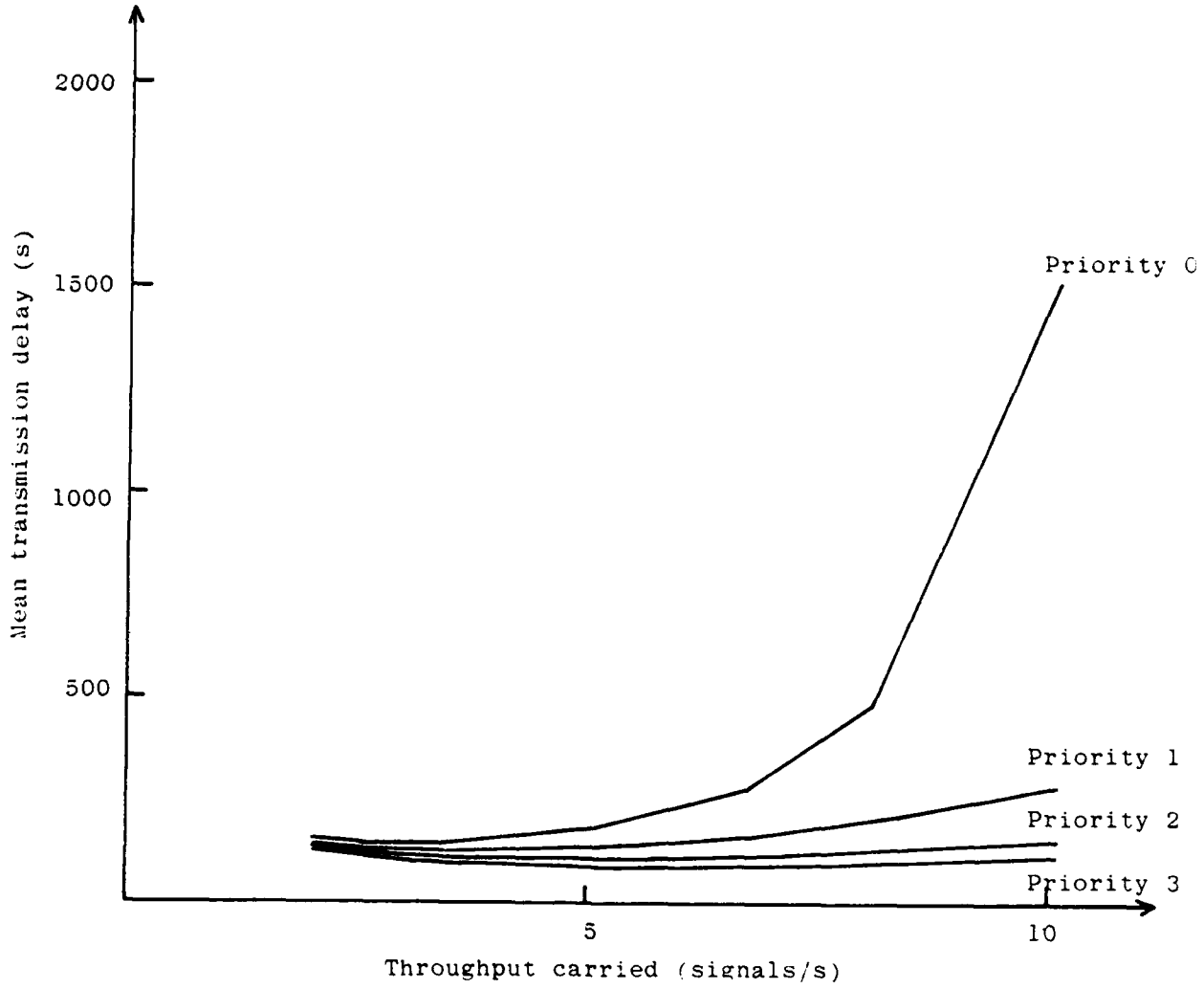


Figure 4.7. Graph Showing Effect of Signal Priority

The send-and-forget nature of CHILL signals requires maintenance of buffers or tables somewhere in the system and the accessing of these, sometimes, by more than one process instance. The models show clearly that performance is heavily dependent on contention for those shared data structures. Contention itself may be influenced by the policies adopted for mutual exclusion between processes during key operations. Relatively minor changes to the locking protocols used were seen to produce dramatic effects on contention levels and hence on system performance.

It appears that the CHILL "open destination" signals pose a severe implementation difficulty. Whilst the solution of model (3) for shared memory gives acceptable results, it cannot be used for non-shared memory architectures and so any implementation for the latter is likely to be inefficient and is perhaps best avoided!

The buffering required for CHILL signals also causes problems. Potentially large numbers of signals must be stored somewhere in the system and, since such storage is likely to be finite in size, overflow can occur. Also, a sender has no assurance of the delivery of any signal sent, since it may remain in the tables for an unpredictable period.

To prevent overflow, some kind of flow control would be desirable. A possible way to provide this is to arrange that a "sender" is held up of the table to which it is attempting to add a signal is full. Sending would be allowed to continue when a vacancy in the data structure appeared. However, this would mean that the SEND operation was no longer always a fast non-wait operation. To increase the security of the signals mechanism, acknowledgements could be used. These could be user provided, by means of explicit SEND operations used to acknowledge a successful RECEIVE, or could perhaps be provided automatically.

Finally, the priority scheme for signals does not appear to be very successful. Since there is no guarantee of delivery time for any signal, the low priority ones may in practice remain in tables indefinitely, incurring long transmission delays. To avoid this problem, it is suggested that only two priority levels be available. These would be defined as "normal" and "emergency", with the latter used very sparingly. In this way, there would be little chance of the normal priority signals suffering excessive delays, yet there would be a faster delivery available to the occasional emergency signal.

The above considerations cast grave doubts over the whole CHILL signals scheme. The impression is of an insecure communications method with large overheads. It is suggested that a message system based on "send-wait" and "receive-reply" primitives, where the sender waits for a reply before continuing, would be better from both a security and implementation viewpoint. Using send-wait, the number of buffered messages is automatically limited to one, so all problems of large buffers and flow control immediately case to exist. It is worth noting that the primitives of Ada, for example, correspond better to the send-wait approach, despite their "procedural" presentation (19). The same philosophy appears in the recently introduced Occam language (20).

5. MICROPROGRAM DEVELOPMENT ENVIRONMENT

5.1 INTRODUCTION

The design of a microprogrammable node for CYBA-M has been one of the objectives of this research project. In order to achieve this end a microprogram development system has been specified and designed to enable the development of a microprogrammed processing element.

5.2 MICROPROGRAM DEVELOPMENT SYSTEM

The major units of the system are indicated in Figure 5.1. The function of each unit has been specified and printed circuit boards have been designed and manufactured (21).

Work has been held up on this project due to the move of Professor E L Dagless to the University of Bristol and difficulty in recruiting staff to complete the project. However, work recommenced in July 1983 when two vacation students were recruited to complete the physical assembly of the development system working under the direction of Dr M Bolton. The following tasks were completed:

(a) Mechanical Assembly

The development system card frame and backplanes were installed. The PCB connectors were mounted and two bus termination circuit boards were designed and manufactured. Power supplies and fan were installed.

(b) Workstation Controller

As initially conceived, the development system would be attached to a PDP-11 host via a DR11-C parallel interface. The present arrangement uses a 68000 single board computer, as the workstation controller, and a VAX 11/750 as the program design system. The VAX runs the microprogram cross assembler and the 68000 the debugger control software.

The 68000 on-board monitor permits the user to communicate with the VAX in a transparent mode.

(c) System Interfaces

The development system interface card had been designed to expect a DR11-C interface. This interface was modified to connect to the 68000 workstation controller. Software was written for the 68000 to permit a user to send and receive data to and from the development system.

(d) Microprogram Assembler

In November 1983, a research student commenced working on the project. To date he has successfully transported the assembler from a PDP-11 to the VAX, and is currently writing the debugging software for the 68000 (in C), together with assembling and testing the remaining cards for the development system.

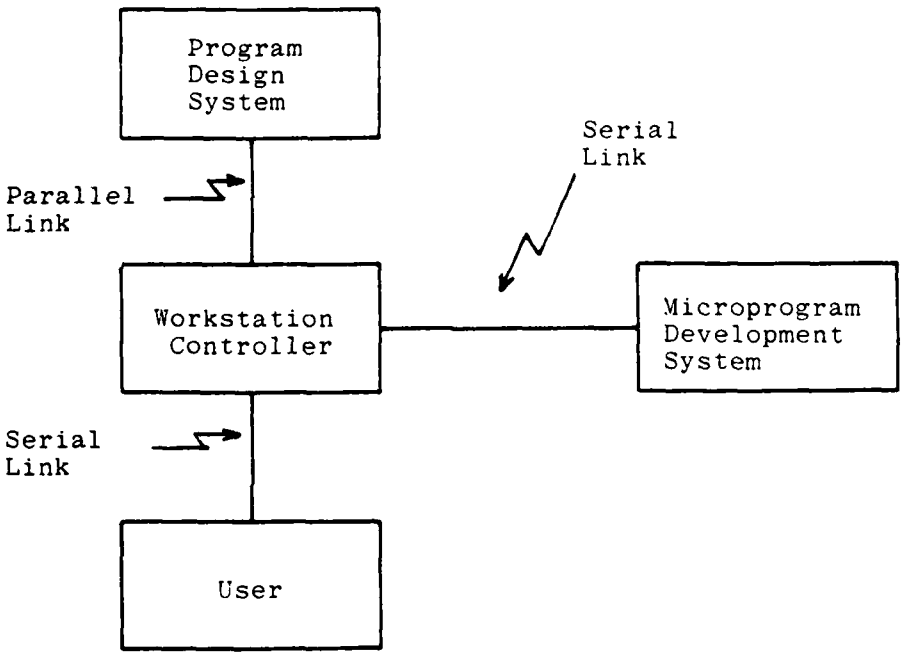


Figure 5.1. Overall Microprogram Development System

5.3 CONCLUSIONS

The construction of the microprogram development system has been delayed for the previously stated reasons. However, it is expected to have a viable development system in the summer of 1984 when research will commence on the design of a microprogrammable mode for CYBA-M.

6. RELATED WORK

In addition to the major aspects of the research program, a related project has been undertaken, by an MSc student, to design and implement an 8086 node to replace one of the current 8080 nodes in CYBA-M.

6.1 DESIGN OF AN 8086 NODE FOR CYBA-M

As part of the overall research project the design and implementation of an 8086 microprocessor node for CYBA-M was undertaken by an MSc research student, G. Rubner (22). The objective was to assess the mechanisms to be employed when upgrading an existing 8080 microprocessor node and to assess the performance of the new node in a heterogeneous processing element environment.

A block diagram of the 8086 node connection scheme is given in Figure 6.1. A standard node interface card (NIC) is provided within CYBA-M for permitting external devices to replace an 8080 node. The NIC presents a standard interface to the external device and also conforms to the access protocols for the local, global and image memories. It was decided to employ the NIC to connect the 8086 to CYBA-M, as the student did not have enough time to design a special purpose interface. The 8086 microprocessor was incorporated in an SDK86 single board processor system. An Intel MDS was used to download ASM86 programs into the SDK86 and CYBA-M. The major part of the design effort was expended in the design and implementation of the interface to connect the NIC to the SDK86 (N861).

A block diagram of the SDK86 and the N861 is given in Figure 6.2. It was decided to map the total address space of CYBA-M (512K bytes) directly onto the address space of the SDK86 (1024K bytes), so that the 8086 system does not have to discriminate between its on-board memory and the CYBA-M memory space. One problem, which was overcome, was the mapping of the 16-bit word length of the 8086 onto the 8-bit word length of CYBA-M. The 8086 can perform byte and word accesses, which meant that the N861 had to perform one or two CYBA-M accesses respectively.

The performance of the system was not as good as expected. A byte access to a CYBA-M memory location was, on average, 2.9 microseconds and a word access 4.9 microseconds (with a 5 mHz 8086 clock). The bottleneck in the system was the NIC, which added an overhead of 1.5 microseconds to each byte access in CYBA-M. However, a special purpose node interface within CYBA-M would remove this overhead time. The overhead was caused by synchronising the operation of two asynchronous state machines, one in the NIC and the other in the N861.

The feasibility of connecting a different microprocessor node to CYBA-M has been demonstrated and experience has been gained in overcoming the problems inherent in interfacing a new microprocessor to an existing multi-microprocessor system.

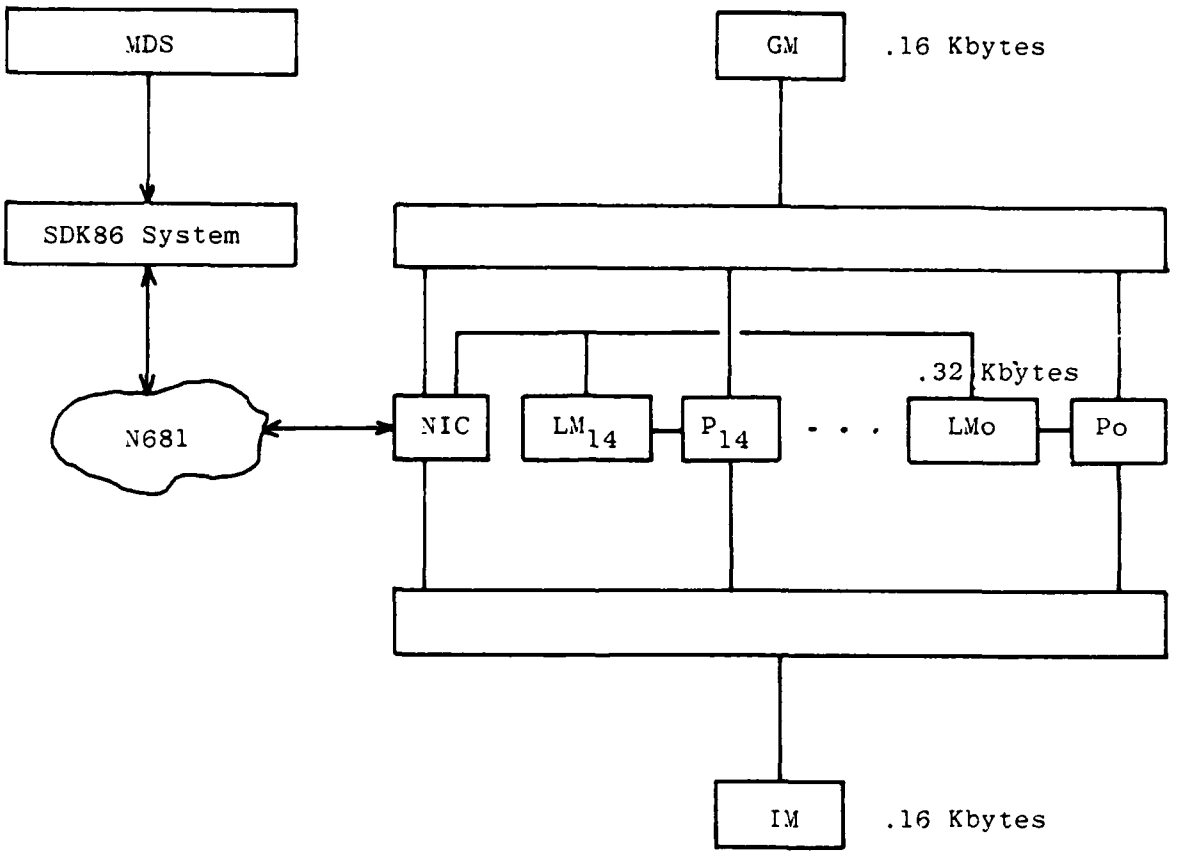


Figure 6.1. Block Diagram of 8086 Node Interconnection System

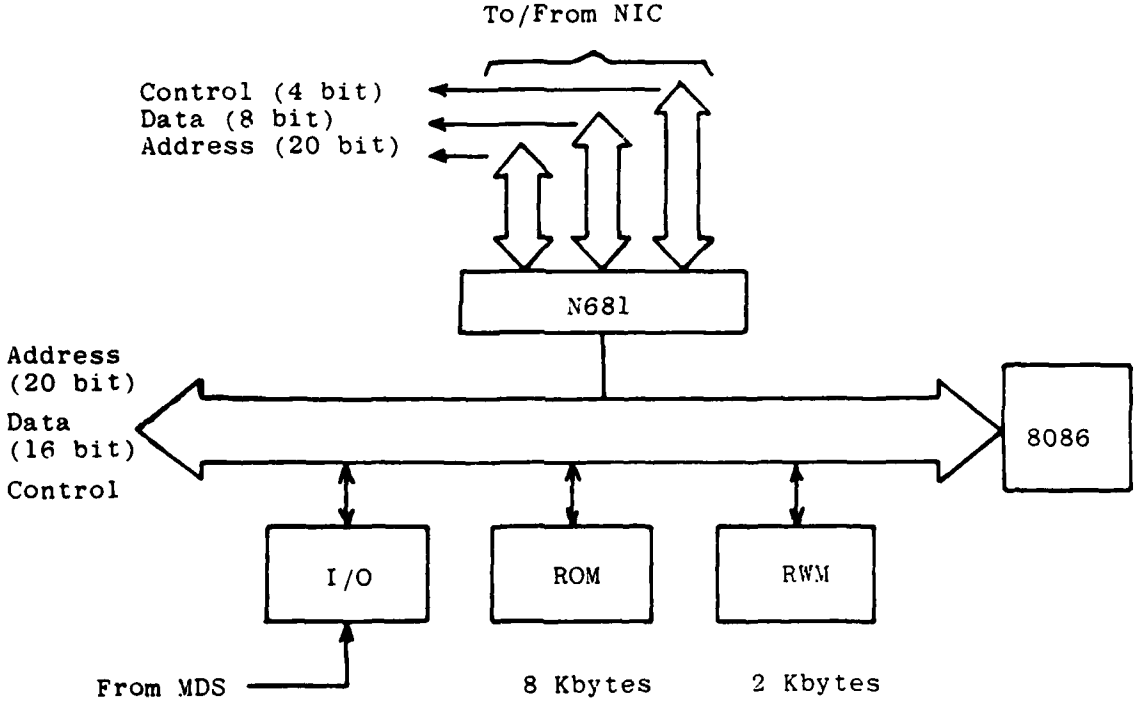


Figure 6.2. Block Diagram of SDK86 and CYBA-M Interface

7. CONCLUSIONS

Multi Processing-element Systems are clearly complex not only in the nature of the interconnection structure but also the complexity of the application to be placed upon them. Whereas concurrency can be simulated on a uniprocessor, it is appreciated that the true concurrency of a Multi Processing-element System is the only way to obtain the correct feel of such a system. Observation of the operation of the direct shared memory system makes it possible to identify the features which are of particular importance. The modelling of the token passing ring highlighted the particular performance parameters which the designer needs to take account of in the implementation of such systems. The experiments on the Multi Processing-element Systems led to the development of the theoretical model which now makes it possible to analyse the performance of the system and to suggest ways in which they may be simulated on a uniprocessor. Experiments on the Multi Processing-element Systems were difficult to conduct. In the early days when simple models were being built and experimented upon,it was quite sufficient to use the assembly code of the microprocessors. However, as it became necessary to extend the scope of the experiment and it was felt desirable to provide an interactive interface for the experiment, then the limitations of assembly code became important. The unidirection ring simulation model described in Appendix A is probably the ultimate which one would attempt to build upon such a system. In future, one would only contemplate modelling complex systems on a multiprocessor if the individual processing elements themselves were better able to support a high level language which in turn would enable the design of the complex model required. Now that there is a better understanding of the theoretical models of such interconnection structures and of the performance criteria to be investigated, it is becoming easier to conduct the simulation experiments on a uniprocessor system able to support the appropriate high level language and to provide the convenient experiment environment. It is recommended that future investigation studies be carried out upon a high performance uniprocessor with a user-friendly environment rather than the development of a multiprocessor system specifically for this purpose.

The direct shared memory system was used successfully to investigate the implementation of the CHILL language. In particular, the CHILL signals were treated in great detail. The limitations of these signals have been clearly exposed and it is suggested that future investigations on real ring systems which address particularly these aspects of the CHILL language should be undertaken.

The use of a multiprocessor to simulate other multiprocessors is not an easy way to proceed, though it may be the only way to observe the true effects of concurrency.

APPENDIX A

UNIDIRECTIONAL RING SIMULATION MODEL

ABSTRACT

Work on the modelling of multi-microprocessor interconnected structures on a Direct Shared Memory (DSM) multi-microprocessor system is being carried out in the Department of Computation, UMIST. The objective of this work is to obtain performance measurements for the modelled structures. Earlier results obtained from modelling a simple ring structure enabled practical experience to be gained in the modelling of such systems and also indicated the problems associated with such modelling. Using this experience, it was decided to write a more sophisticated simulator, so that the earlier problems could be overcome, and more accurate results obtained.

INTRODUCTION

CYBA-M (23) (Figure A.1) is a Direct Shared Memory (DSM) multi-microprocessor system, which has fifteen microprocessors available for use. These microprocessors (Pnrø-14 in Figure A.1) are Intel 8080s and each has its own local memory (LMø-14 in Figure A.1) of 32K bytes. This combination of an 8080 microprocessor and a local memory is termed a processing element, and communication between processing elements is accomplished using the Global Memory, which has 16K bytes of storage. The various program modules are loaded into the appropriate memories of the processing elements under control of EMU (24), an operating system which runs on CYBA-11*. The system provides a suitable vehicle for modelling other microprocessor interconnection structures, since the Global Memory can act as the connection medium, in place of the cabling etc. of the systems under study.

PREVIOUS WORK

A modelling system developed by Kille (25) enabled the Global Memory of CYBA-M to act as a series of communications buffers between processing elements on CYBA-M, in such a manner that these buffers can represent interconnects between nodes for loop, complete and star facilities felt to be necessary for detailed investigations and a course of work was embarked upon by Ainscough to adapt these modules so as to create a software local network modelling capability, with sufficiently detailed monitoring capability that the system would suit our requirements.

A simple ring structure is shown in Figure A.2, where each node of the ring uses two processing elements on CYBA-M. The host processor is reserved for user (applications) processes, and has a simple interface to the comms (communication) processor, which interfaces to other nodes in the ring. An experiment was performed by Rivkin (26) using the modified version of Kille's system to implement the ring of Figure A.2.

---

* CYBA-11 is the name given to a PDP which is dedicated to CYBA-M.

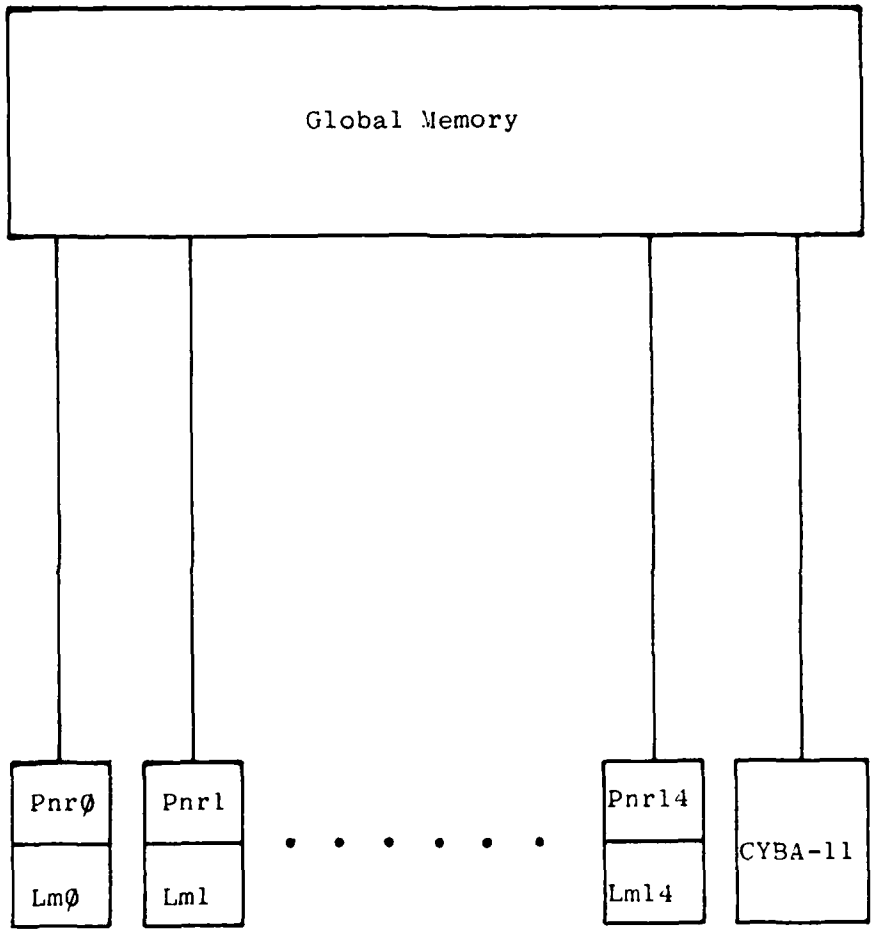


Figure A.1. CYBA-M Multimicroprocessor System

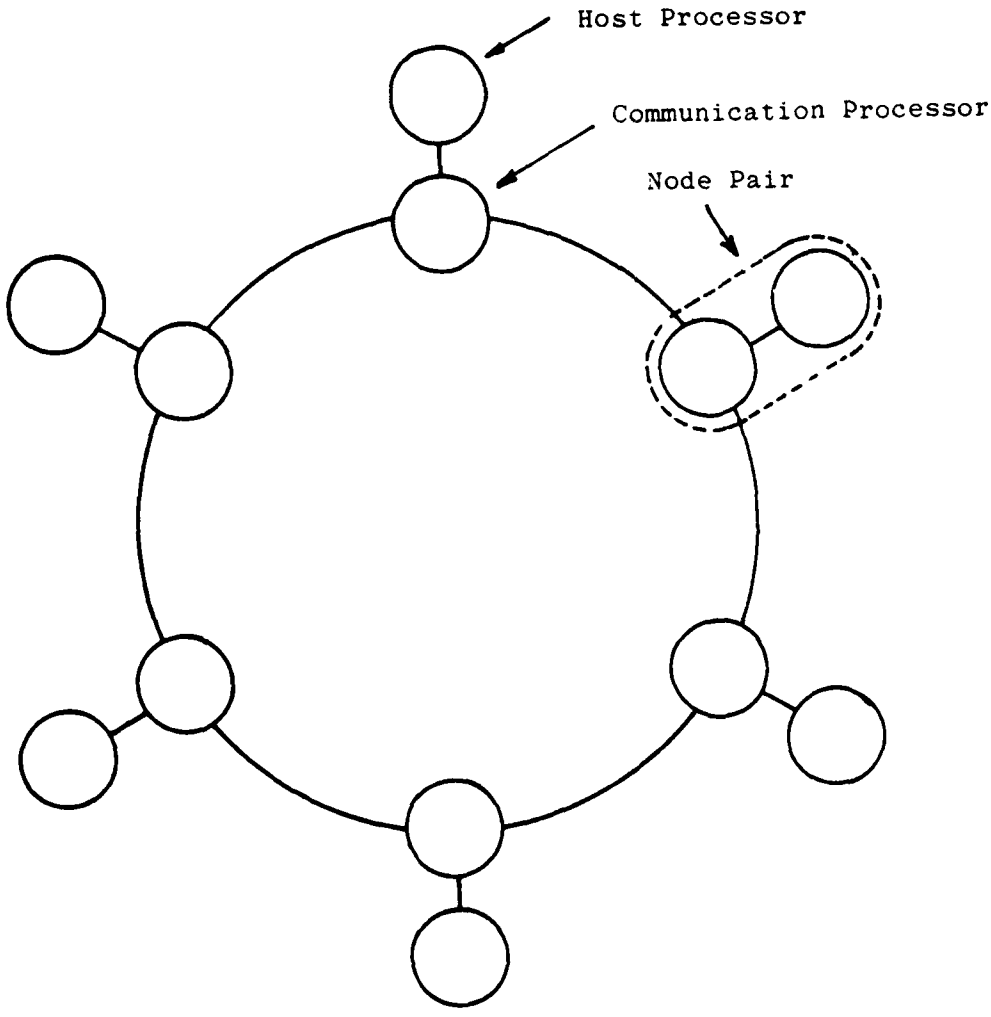


Figure A.2. Simple Ring

In this experiment, one source node sent messages to the next (destination) node, so that only two ring nodes were used in the simulation and this represented the simplest case of sending and receiving a message. Many simulation runs were performed, such that on each run the time between sending messages was altered, but was set at a constant for any given run. The messages were monitored at various points in the node, and in particular the times from when messages were initiated until they were received by the destination, and the times between subsequent messages being started (i.e. the queueing times in the sending node) were monitored. In order to avoid problems due to "starting up", on each run 1000 messages were allowed to be sent and then the next 10 messages were monitored. The mean values of the times monitored from these 10 messages were used to obtain the results shown in Figure A.3.

The results were presented in a form proposed by Barton (27). Calls are made to the communication processor for messages to be sent, each call lasting for time "e", after which a further time "a" is spent in the application processor before the next call to the communication processor is made. Plotting $1/(a + c)$ versus $1/a$ represents the number of messages per unit time (i.e. throughput carried) versus throughput demanded (i.e. throughput carried when $C = D$ - the idealised system).

The performance of the idealised system is shown in Figure A.3 by a 45 degrees straight line. The modelled system tracks approximately 3% below the ideal until the demanded throughput rate reaches approximately 18,000 requests per second, when there is a dramatic fall in the throughput carried to approximately 40% below the ideal. As the demanded rate increases there is a violent oscillation in the carried throughput due to synchronisation of the communication processors. The oscillations abate and the carried throughput tracks at approximately 40% below the ideal.

The angular plot which resulted from the experiment is not a feature of Kille's model, however, but is due instead to the various processes in the model synchronising and then de-synchronising as the varying source arrival rate occasionally causes the processors to cooperate well on the task of modelling, due to the various pieces of code having path lengths which correspond in such a manner that a form of "beating" takes place. This phenomenon was also encountered by Barton, who then solved the problem by using a Poisson distribution of the source arrival rate having a mean value equal to the desired request rate. The effect of this was to randomise the request rate and so break up any potential process synchronisation.

It was stated earlier that Kille's original set of modules had been adapted for use in this particular application. Inevitably then, since the modules had not originally been written with this application in mind, it could not reasonably be expected that they would exactly meet our requirements in total. However, adapting this system did provide valuable experience, and helped to identify more clearly our requirements for a simulator written specifically for this application. One of the main problems encountered with Kille's model was that timings from the system were obtained by using a code-generated timing loop, which timed the various paths of code in the model. This method, together with the fact that Kille moved messages around in sixteen byte blocks examining buffers on a round-robin basis, meant that the timings were not accurate enough for our purposes.

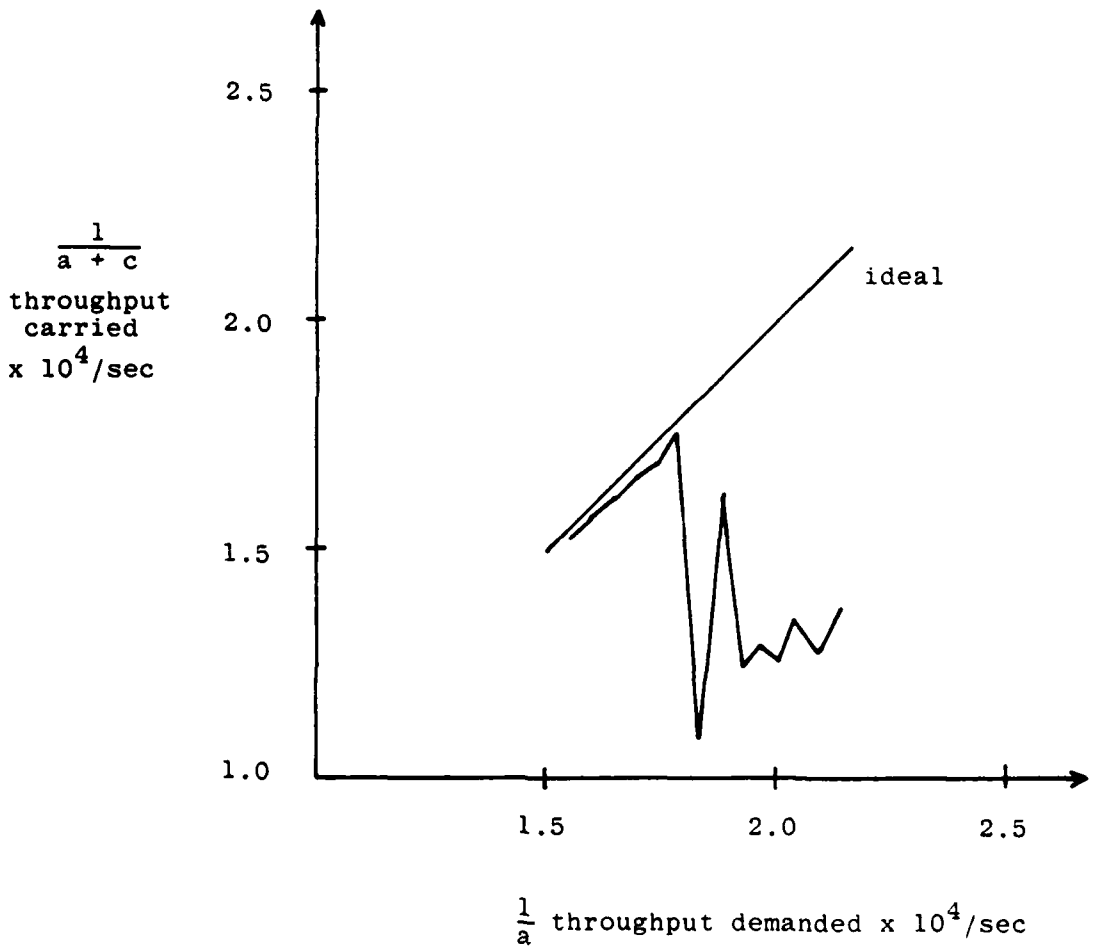


Figure A.3. Performance Results

SIMULATION REQUIREMENTS

It became apparent that what was required, in order to improve the accuracy of the timings obtained from simulation, was that the simulations should be rigidly tied to an accurate simulation clock. Initially, it was decided to allow a one-byte movement of data in each clock period of the transmission, but eventually it is intended that a single bit movement of data should occur in each simulation clock period. Such rigid control of the movement of data with regard to time would also prevent the synchronisation phenomenon of Kille's model.

A further requirement was that the simulator should be designed in such a way that it would be sufficiently well-structured and flexible, so as to allow it to be easily modified to include future, but as yet undecided, topologies and protocols. It was decided to write the simulator in CYBA-Pascal, a version of Vrije Pascal which had added communication primitives. Tx (transmit) and rx (receive), which could be used in complementary pairs. These primitives allowed for the transmission and reception of variables between concurrent processes on physically different processors, by making use of the global memory, and they also enabled these processes to be synchronised by rendezvous, since the first instance of either tx or rx resulted in the process in which that instance occurred being forced to wait for a matching instance of the complementary primitive to occur in another process.

It was decided that the first type of ring to be studied was a unidirectional token-passing ring, and that the transmission format used should be HDLC. Once this ring had been studied, it was intended that other ring protocols and data formats should be added to the simulator to build up a comprehensive repertoire, so that a variety of structures could be studied under various traffic conditions.

In order to simulate the simple ring of Figure A.2, buffers must be introduced into the model (Figure A.4(a)) to allow for the transmission of data around the modelled structure. Access to these buffers must be strictly controlled using the synchronisation primitives, so that during one clock period, data progresses through each node, from input buffer (phase (i)) through to the output buffer (phase (iv)), and where necessary, undergoing movements through phases (ii) and (iii). An individual node is shown in Figure A.4(b) where the input/output buffers have shared ownership between neighbouring communications nodes, such that the output buffer of one node is the input buffer to the successor of that node.

In terms of the capabilities of the simulator on CYBA-M, this means that it is possible to simulate rings of up to seven nodes, since two PnVs* are required for each node (for the host and comms) making a total of 14 PnVs required for the nodes. One PnV is required to connect to CYBA-11, so that we have one spare PnV left over. This spare PnV is used as the "monitor", and this is used to control the simulation.

Since inter-processor communication is performed on CYBA-M via the global memory, then the buffers above must also be implemented

---

* A PnV is the number given to a virtual processor on CYBA-M, and this virtual processor maps on to a real processor (Pnr).

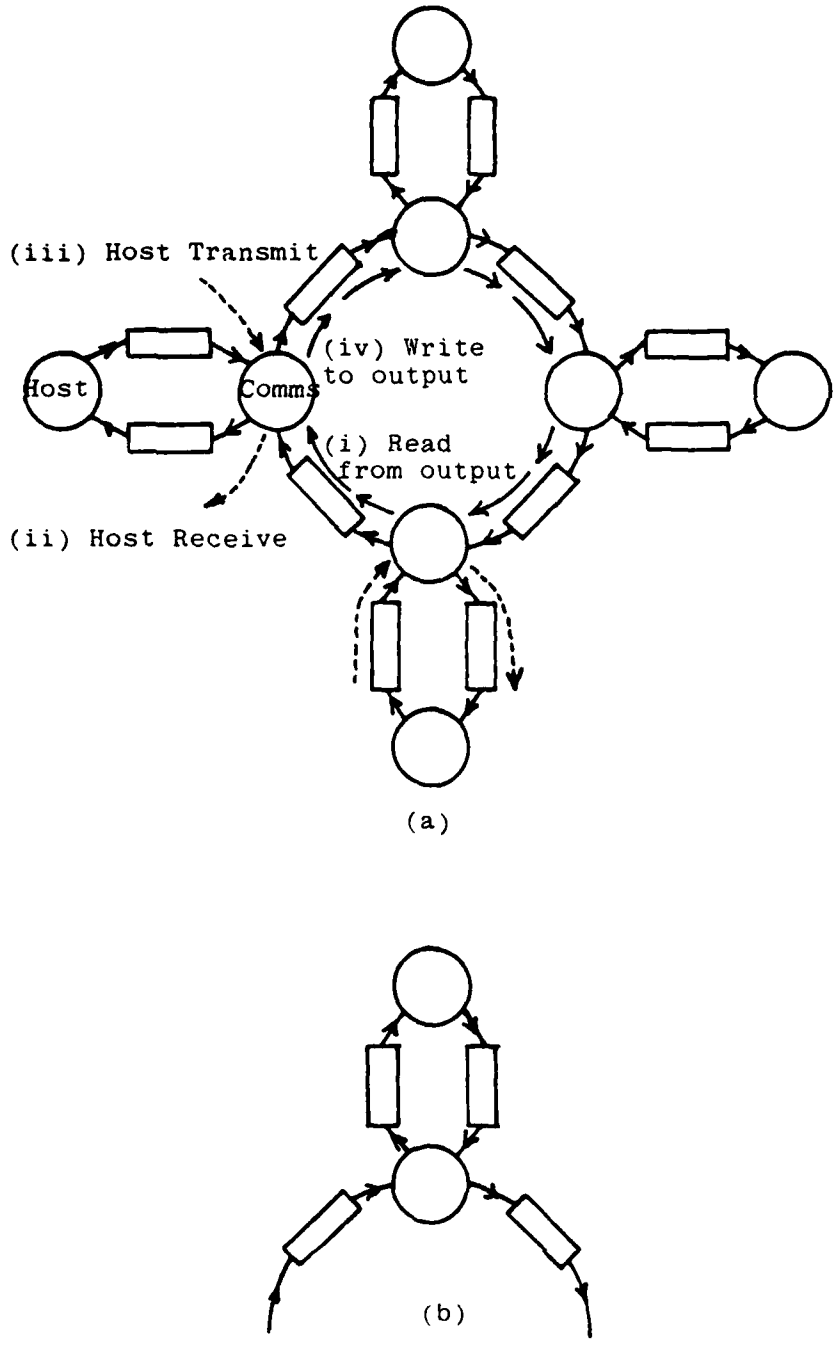


Figure A.4. Basic Ring Simulation Requirements

in global memory. Therefore, in order to create space in global memory, it is only necessary to define a pointer to a data item and then "new" the item onto the heap. In order to model the ring, then, it was necessary to define a global datastructure.

DEFINING A DATASTRUCTURE

In addition to controlling the simulation, the monitor PnV is also responsible for building the simulator data structure in global memory. The monitor interactively determines how many nodes (minimum 2, maximum 7) are to be simulated, and then builds the data structure shown in Figure A.5(a), creating a set of records for each host/comms pair. The ring buffers for comms to comms data transmission are also established at this time. The record structure for a host/comms pair is shown in detail in Figure A.5(b). It can be seen that the structure contains information pertinent to that node pair in particular, but it also establishes the position of the pair in the ring, such that whereabouts of the next neighbour pair is known, so that by using the next neighbour pointers, the ring is completed.

INITIALISING THE DATA STRUCTURE

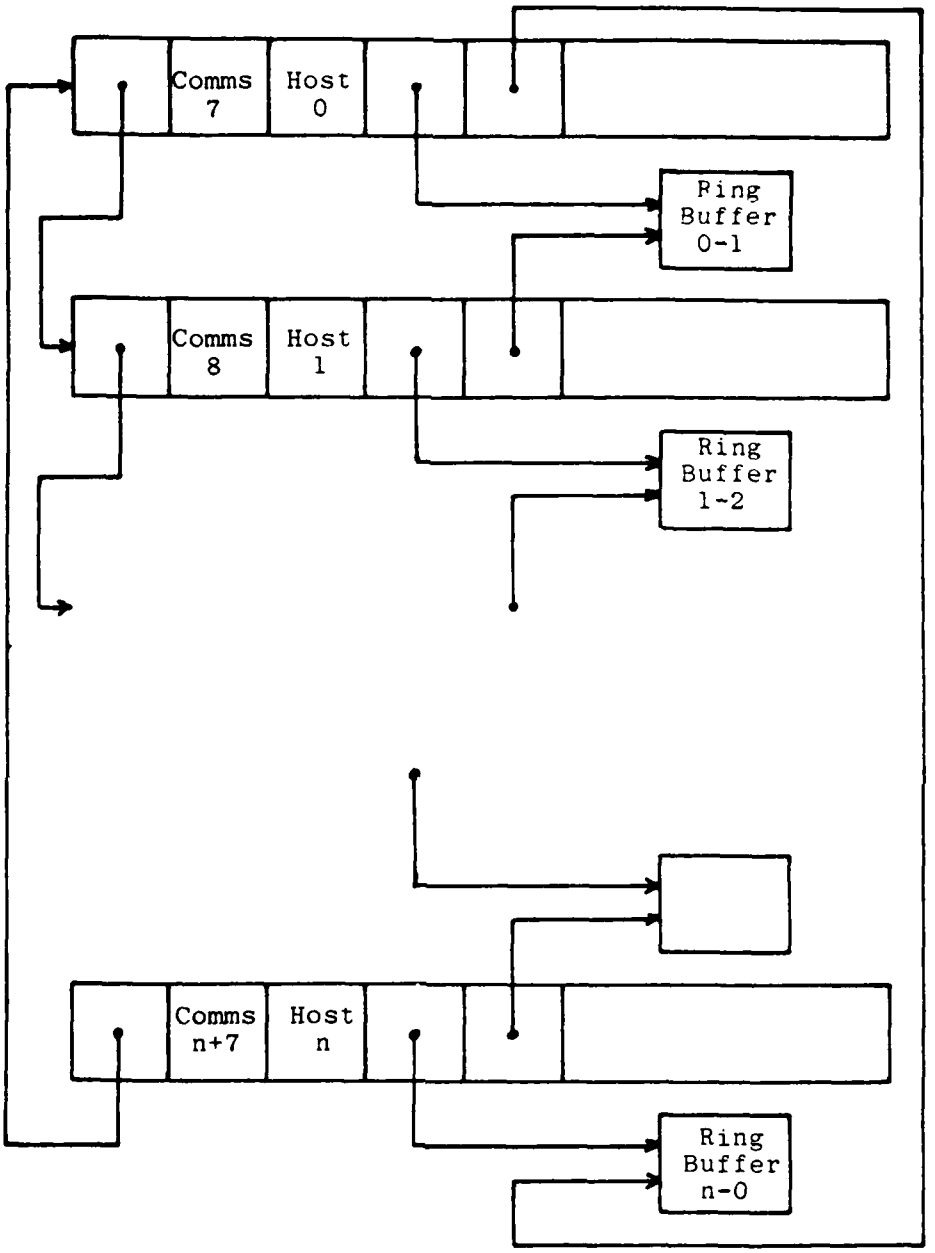
After building the data structure, the monitor then interactively determines which nodes are sending messages, and the destinations of these messages, together with the message sizes in bytes, and the number of messages to be sent. This information is loaded into the data structure to initialise it. The buffers used for the transmission of data are initialised with "flag" bytes to signify that they are not yet carrying message data, and the input buffer to the node containing host Ø is loaded with the "token" byte. During the course of the simulation, monitoring information concerning the state of a node with regard to message passing and fault conditions, is maintained, as is information concerning the sending rate, and number of messages still remaining to be sent from that node. The process in the host of a node is deemed to be active if the send-mode record indicates that the host still has messages which are required to be sent. However, any mode, even with a process which is not active, is capable of receiving messages.

DISTRIBUTION OF THE DATA STRUCTURE

When the data structure has been initialised, the monitor uses the tx primitive to transmit pointers to the PnVs of each host/comms pair, so that the host and comms processes can access their own particular part of the data structure, as shown in Figure A.6. Thus a node pair may read and write to its own message buffers, and also to the ring input and output buffers to which it is immediately connected. It can also update the send-mode and monitoring-information records. Since the monitor process has knowledge of the whole data structure, then it is able to have access to the whole structure.

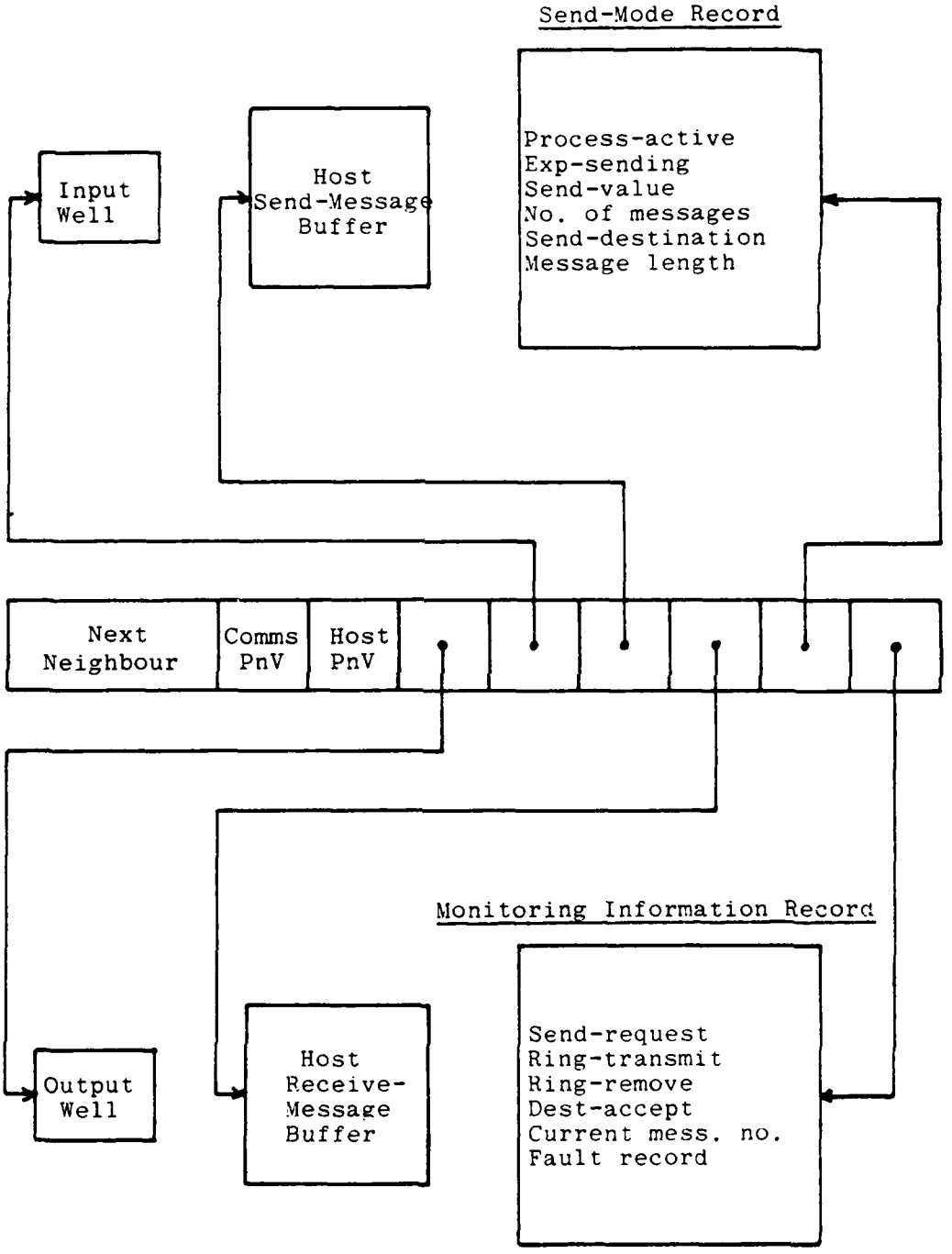
MESSAGE DATA PATHS BETWEEN HOST AND COMMS

Queues are used in the local memories of the comms nodes to decouple the host and comms processes during the sending and receiving of messages. Thus, as shown in Figure A.7, the host writes a message to the host send buffer, and the host process is then free to continue. The comms process, upon detecting that the host send buffer is full, unloads this buffer into the back of the output queue, and frees the buffer. Eventually, this message reaches the front of the queue and is then formatted in HDLC format and loaded into the output buffer. The output buffer is then unloaded, a byte at a time in each simulation



(a)

Figure A.5. Global Memory Data Structure


Send-Mode Record
Process-active
Exp-sending
Send-value
No. of messages
Send-destination
Message length
Input Well
Host Send-Message Buffer
Next Neighbour
Comms PnV
Host PnV
Monitoring Information Record
Send-request
Ring-transmit
Ring-remove
Dest-accept
Current mess. no.
Fault record
Host Receive-Message Buffer
Output Well

(b)

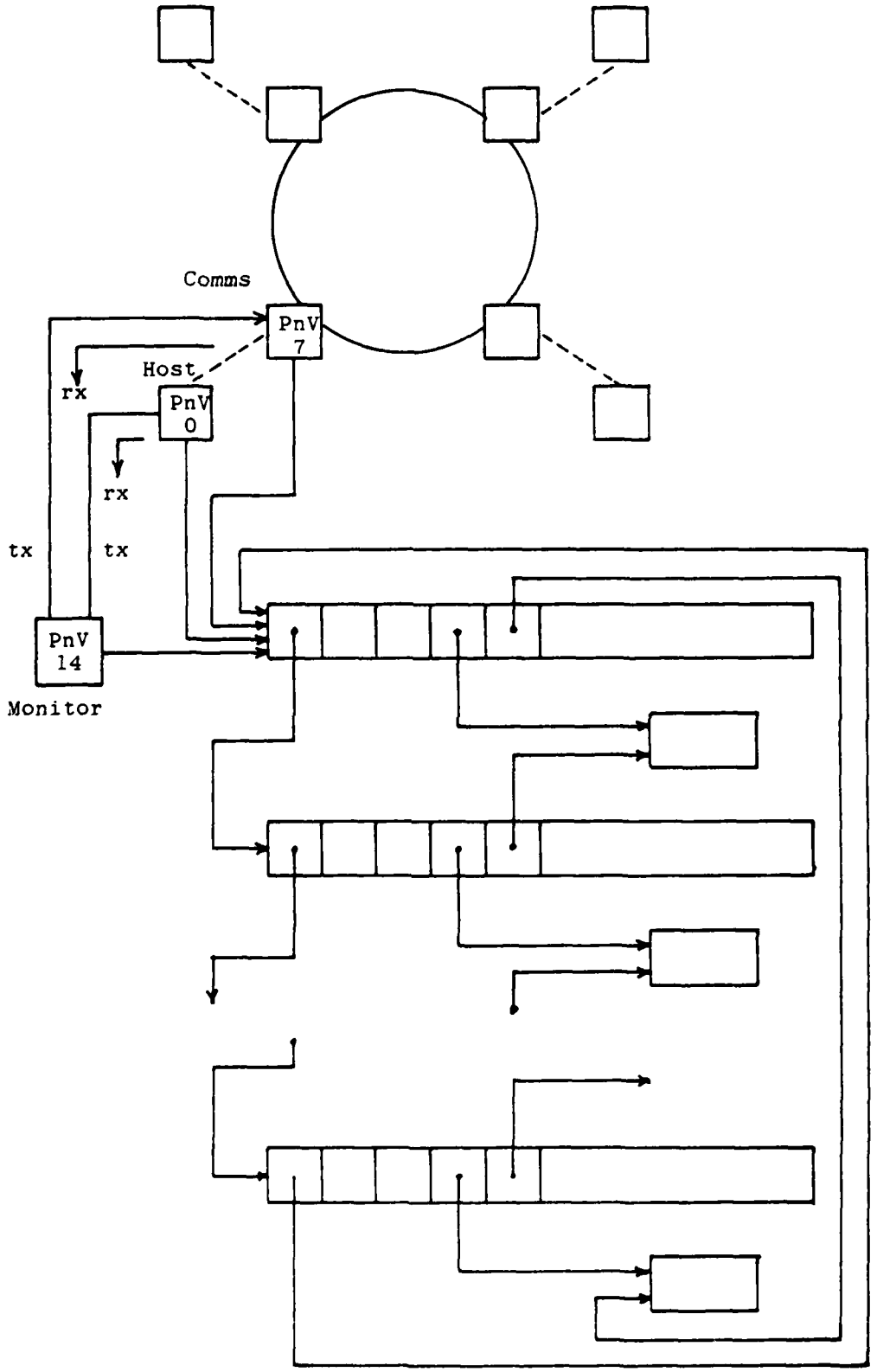


Figure A.6. Distribution of the Global Data Structure

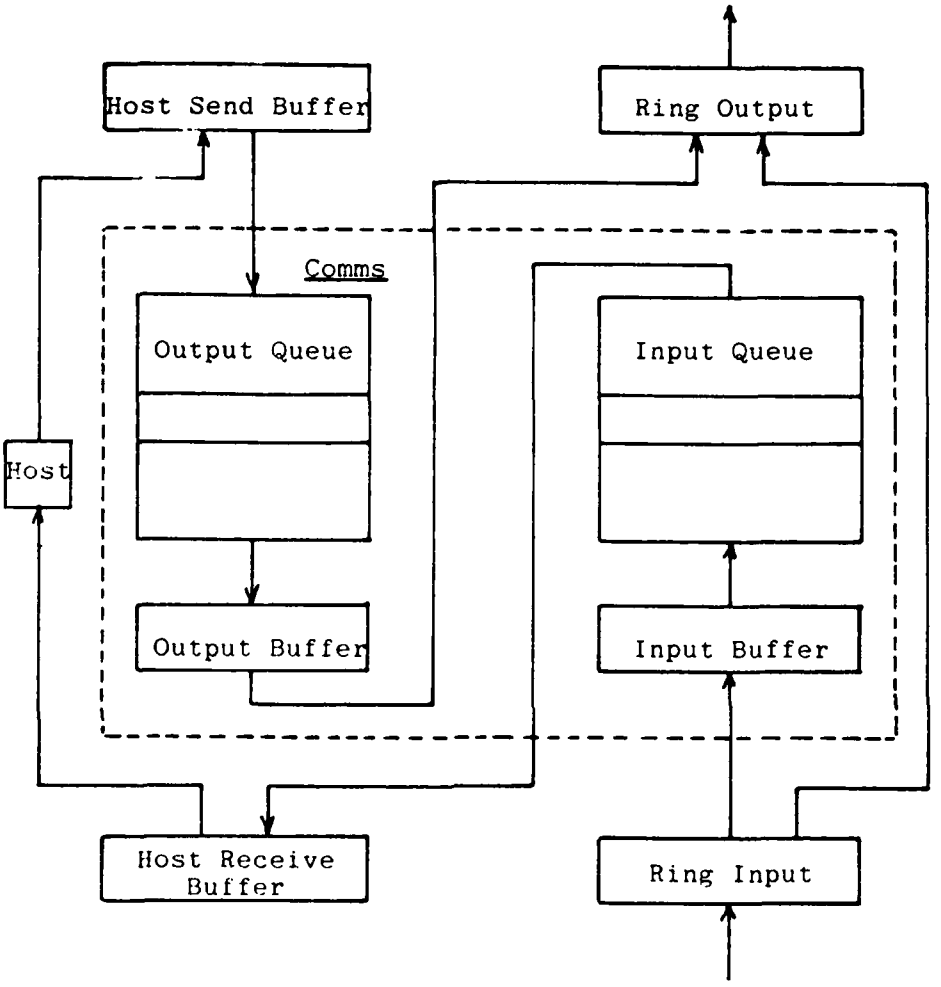


Figure A.7. Node Data Flow

clock period, into the ring output buffer. In the absence of any message for transmission, flag bytes are placed in the ring output buffer in each clock period.

Reception of a message is similar, in that message bytes are taken from the ring input buffer of the destination node during each clock period. The bytes are then loaded into the input buffer, and then, after being de-formatted, are loaded into the end of the input queue. The message at the top of the input queue is then written into the host receive buffer. Upon detecting this buffer full, the host process removes the message and frees the buffer. If a node, at a given point in time, is neither sending nor receiving data, then it bypasses any data arriving at its node until such time as a message arrives having that node as the destination or the host in the node wishes to send a message. However, even though a node may wish to send a message, it cannot do so until that node receives the token. The token is always circulated at the end of a message or group of messages, so that a node must first absorb or bypass any messages arriving, in order to obtain the token and so gain the required permisssion to send a message. After transmitting its message, the node then regenerates the token so that it can be passed on to its next neighbour.

CAPABILITIES OF THE SIMULATOR

The simulator can send messages from any node to any other destination node, and these messages may be sent at a constant rate, or at a Poisson rate about a given mean rate. Also, the length of the message in bytes can be varied up to a maximum of 256 bytes and can be different in each node, as can the sending rates.

It is also possible to measure the performance of a particular target system. If it is known that the host processors of the target system have an average instruction time of say, x μseconds, then a Poisson distribution of instruction wait times with a mean value of x is loaded into global memory. These times are then scaled so that they are in terms of simulation clock units, so that, for example, if the target ring speed were to be, say, 10 mbit/sec, then a one-byte movement would take 800 μseconds of real time, so that the simulation byte clock would achieve a one-byte movement of data which would have taken 80 μseconds in real time. Thus, if the target processor speed were, say, an average instruction time of 4 μseconds then the instruction wait times would have to be multiplied by 5, so that they would then be in clock units, such that 5 clocks = 4 μseconds of real time. When the simulator is eventually modified to have a bit clock, that basic clock unit will then be 100 μseconds so that scaling will be simplified. At each clock the current instruction wait time is decremented by 1 until it is zero, at which point a target instruction is said to have been obeyed. The next value of instruction wait time is then loaded for use in the following clock periods. If an experiment is to be performed in which it is wished to determine ring traffic behaviour when a node sends a message on average, after every n target instructions, then a second wait counter, the send-wait counter, is decremented by 1 each time the instruction wait counter reaches zero as shown in Figure A.8. Thus, if the send-wait counter is loaded with a value of n, when this value reaches zero, n target processor instructions would have been obeyed, and the host processor then sends a message, and re-loads the send-wait counter once more with the value n. Obviously, it is possible to obtain processor-independent timings by specifying a hypothetical target processor with a convenient value for the average instruction time.

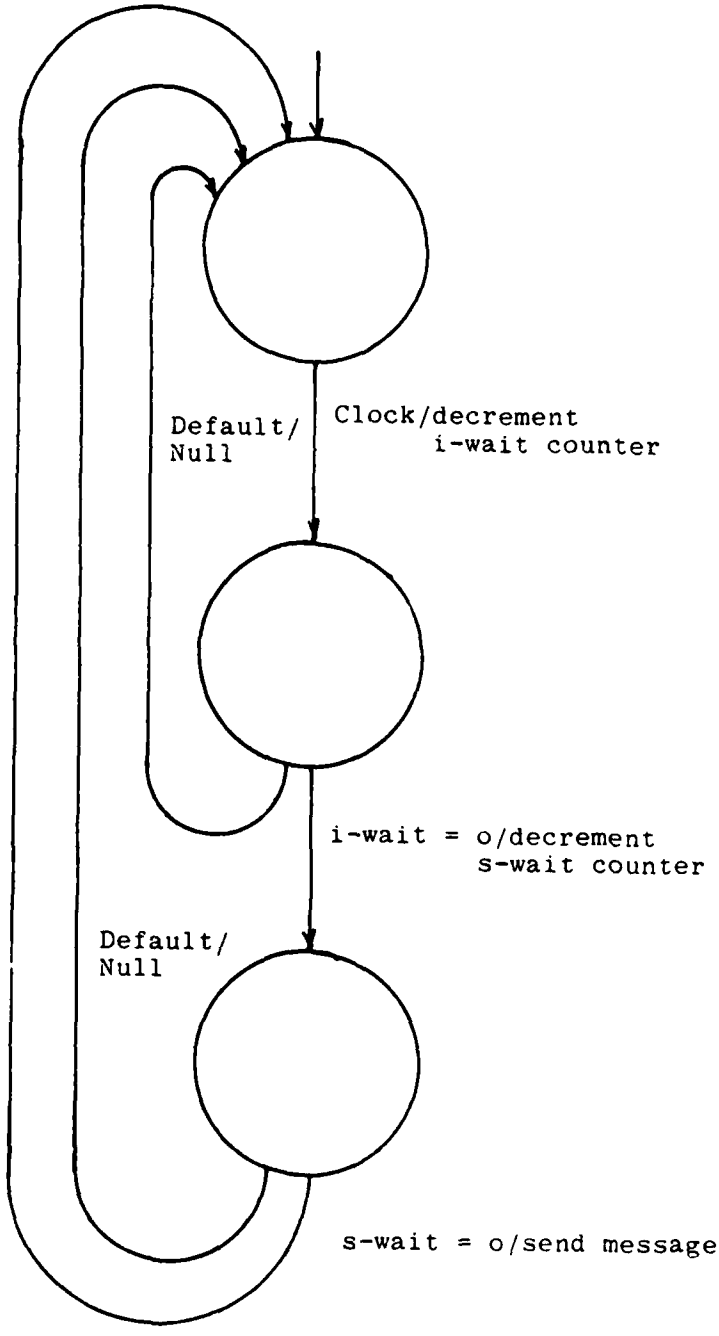


Figure A.8. Injection of Host Messages

CONTROL OF THE SIMULATOR

Once the requirements for a particular simulation run have been established by the monitor during the initialisation phase, the simulation begins and the behaviour of the monitor, comms and host processes is rigidly controlled by finite state machines (FSMs), as shown in Figure A.9). When the monitor issues a clock pulse, the clock is transmitted, using the tx primitive, to all the comms processes. The comms processes then use tx to start their own host process FSM. The host/comms pair FSMs then perform a one-byte data movement within their node, subject to the constraints necessary for a token passing ring, and synchronising their activities with the tx, rx primitives. When a host has completed its tasks for the current clock period, it signals the fact that it has finished to its comms process, and the comms process, in turn, signals to the monitor that it too has finished. After all the comms processes have informed the monitor that they have finished, then the monitor performs its activities, i.e. scanning for any faults, scanning the monitor records, and then updating the simulation clock, at which time the sequence of activities begins again.

FUTURE WORK

Once the simulator has been completely tested, and results have been obtained for the token passing ring, then other structures will be investigated. The host/comms interface will also be enhanced to be consistent with the OSI Reference Model. It can be seen in Figure A.10(a) that, by introducing an extra next neighbour field into the data structure, such that this forms a ring in the opposite direction to the original next neighbour field, and creating an extra set of ring buffers, it is possible to simulate the behaviour of a bidirectional ring. The queueing structure in the local memory of the comms nodes can be kept unchanged, to decouple the host and comms processes, and only the rules relevant to the particular protocol to be studied need be changed.

Bus structures can also be studied as shown in Figure A.10(b), by using the same data structure as for bidirectional rings, with the exception that the ring is left unclosed. Thus data being sent from a source node can move either left or right in the structure, as appropriate, to reach the destination node. Obviously, the rules for arbitration for control of the bus would take the place of the rules relating to the control of the bidirectional ring.

CONCLUSIONS

A simulator has been described which is capable of concurrently simulating the behaviour of traffic on various network topologies, using the global memory of CYBA-M to connect the processors of these structures. It has also been shown that it is possible to tailor the simulator to model a particular target system. Whilst it is true that the processing power of CYBA-M is limited by the fact that Intel 8080 microprocessors are used, the true value of simulating on such a system is that a problem which is concurrent in nature can be solved in a concurrent manner. It is also true that such simulations could be performed on a uniprocessor, however, it is believed that such a task would be more difficult, since there would also be the added requirement of coordinating the process changes in such a manner that data flowed in the correct sequences around the structure.

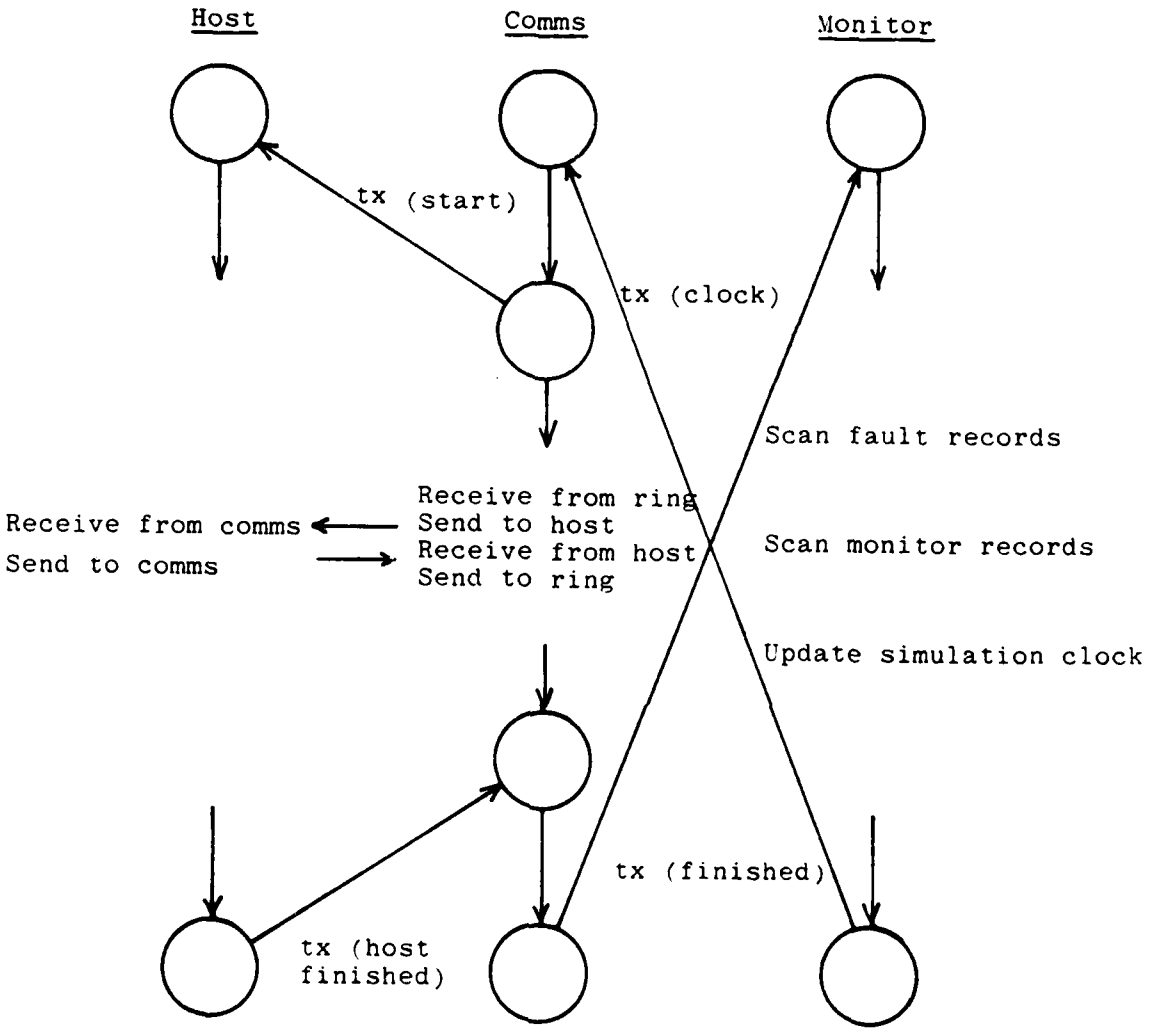
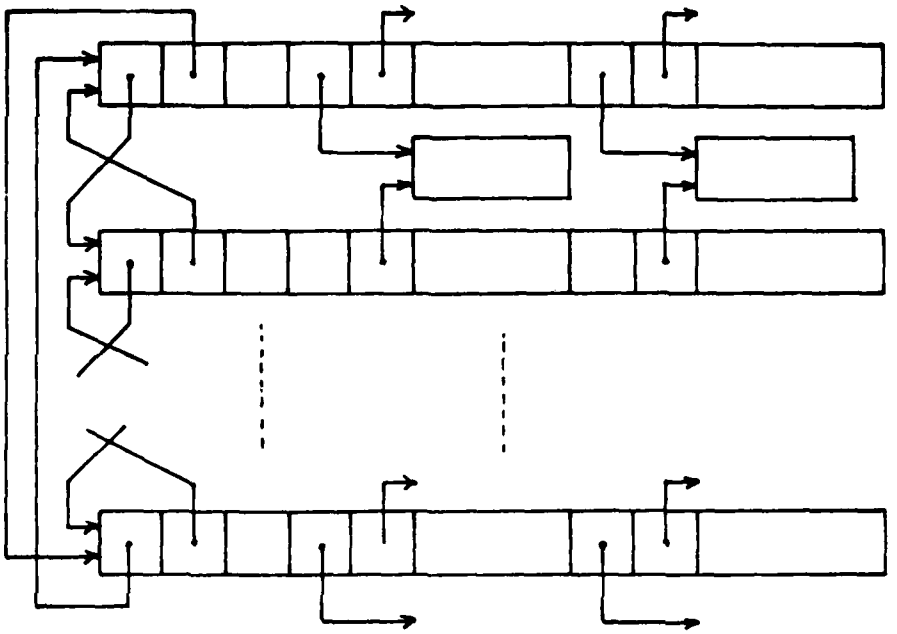
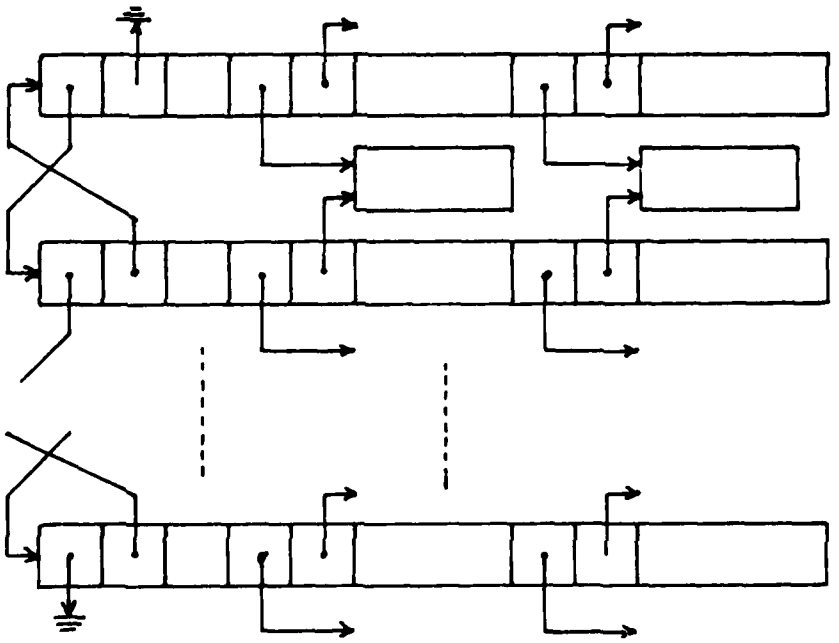


Figure 4.9. Finite State Machine Control



(a) Bidirectional Ring



(b) Bus Structure

Figure A.10. Alternative Configurations of the Data Structure

If, though, one really wished to use a very powerful uni-processor, then if the topology were first simulated on CYBA-M, then all that would be required would be to modify this version of the simulator to include process changing, thus simplifying the uni-processor task. For example, such an implementation could be accomplished under UNIX, using fork to create extra instances of the processes, and pipes to pass information between the processes and to perform synchronisation.

APPENDIX B

THEORETICAL MODEL OF A BIDIRECTIONAL TOKEN PASSING RING

The main characteristic of a bidirectional token passing ring is that it essentially consists of two independent token passing rings. In the first ring, ring A, the token flows in a clockwise direction and in the second ring, ring B, the token flows in an anti-clockwise direction. Both rings operate in an identical manner to a unidirectional token passing ring. The only problem to be resolved is the case where both tokens appear in the same node at the same time - a "token clash".

A number of alternative solutions exist:

(a) If the node has a message to transmit, it

  (i) Selects the ring which gives the shortest path length to the destination node.

  (ii) Arbitrarily selects ring A or ring B.

  (iii) Selects ring A then ring B in a fixed order.

In all cases the other token may be relayed directly to the next node or retained in the node until a new message is ready to be transmitted

(b) If the node does not have a message to transmit, it can

  (i) Relay both tokens

  (ii) Relay one token

  (iii) Relay no tokens

The chosen alternative will affect the ring performance. In order to maintain throughput, the token(s) should not be retained in the node. Thus, a simple theoretical model considers the bidirectional ring as being two independent unidirectional rings. Therefore, the theoretical model developed for the unidirectional ring can be extended to the bidirectional ring.

The bidirectional model is based on the assumption that a bidirectional ring can convey twice as many messages in a unit time than a unidirectional ring. Therefore, the ring utilisation factor, R, should lie in the range 0 to 2. Hence, the stability criterion for the bidirectional ring is:

$$\sum_{\substack{i=0 \\ i \neq j}}^{i=N-1} U_i + \frac{N + S_j}{D_j} \leq 2 \quad \ldots\ldots \text{(B.1)}$$

For all D_i, $0 \leq i \leq N-1$, $i \neq j$: $\frac{1}{D_j} > \frac{1}{D_i}$

The corresponding mean scan time (MST) for the bidirectional ring can be calculated from:

$$MST = \frac{N + S_j}{2 - \sum_{\substack{i=0 \\ i \neq j}}^{i=N-1} U_i}$$

Similarly, for an unstable bidirectional ring:

$$\text{Min (Tsat)} = \text{trunc} \left[\frac{\text{Tsat}}{D_j} \right] \qquad \ldots\ldots \text{(B.3)}$$

$$\text{Mout (Tsat)} = \text{trunc} \left[\frac{\text{Tsat}}{\text{MST}} \right] - 2 \qquad \ldots\ldots \text{(B.4)}$$

$$Q = \text{Min (Tsat)} - \text{Mout (Tsat)} \qquad \ldots\ldots \text{(B.5)}$$

Simulation experiments, similar to those performed for the unidirectional ring, showed a close correlation between the simulation model (now modified for a bidirectional ring) and theoretical model results (12). However, both models will be modified in the future to cater for the alternative viable solutions to token clash problems.

REFERENCES

JOURNAL ARTICLE:

1. Dagless, E.L., Edwards, M.D., and Proudfoot, J.T., "Shared memories in the CYBA-M multimicroprocessor." IEE Proceedings-E, 130, 4, 1983, pp. 116-124.

JOURNAL ARTICLE:

2. Kaye, A.R., "Analysis of a distributed control loop for data transmission." Symposium on Computer Communications, Networks and Teletraffic, Brooklyn, USA, April 1972, pp. 47-58.

JOURNAL ARTICLE:

3. Yuen, M.L.T., et al, "Traffic flow in a distributed loop switching system." Ibid., pp. 29-46.

JOURNAL ARTICLE:

4. Robillard, P.N., "Analysis of a loop switching system with multirank buffers based on a Markov process." IEE Transactions on Computers, COM-22, 11, 1974, pp. 1772-1778.

CONFERENCE PROCEEDINGS:

5. Dagless, E.L., "A multimicroprocessor: CYBA-M." *Information* Processing 77, B. Gilchrist (ed.), IFIP/North-Holland Publishing Co., 1977.

JOURNAL ARTICLE:

6. Burkimsher, P.C., "EMU: A multiprocessor software debugging tool." Software & Microsystems, 1, 2, 1982, pp. 41-47.

REFERENCE BOOK:

7. High Level Data Link Control Procedures, Part 1: Specification for Frame Structure, British Standard 5397, 1981.

BOOK:

8. Kleinrock, L., Queuing Systems, Vol. 2: Computer Applications, John Wiley & Sons, 1976, pp. 56-62.

CONTRACTOR IN-HOUSE REPORT:

9. Houshmand, H.S., An analysis of the traffic flow in a token passing ring, Department of Computation, UMIST, 1983.

UNPUBLISHED DISSERTATION:

10. Alreja, S.K., Performance Evaluation of a Token Passing Ring CYBA-M, MSc Dissertation, University of Manchester, 1983.

JOURNAL ARTICLE:

11. Hayes, J.F., and Sherman, D.N., "Traffic analysis of a ring switched data transmission system." Bell System Technical Journal, 50, 9, 1971, pp. 2947-2978.

IN-HOUSE REPORT:

12. Houshmand, H.S., An analysis of the traffic flow in a bidirectional ring, Department of Computation, UMIST, 1984.

JOURNAL ARTICLE:

13. Stuck, B.W., "Calculating the Maximum Mean Data Rate in Local Area Networks." Computer, 16, 5, 1983, pp.72-76.

JOURNAL ARTICLE:

14. Bux, W., "Local Area Subnetworks: A performance comparison." IEEE Transactions on Computers, COM-29, 10, 1981.

REFERENCE BOOK:

15. CCITT Study Group XI, Introduction to CHILL, May 1980.

REFERENCE BOOK:

16. CCITT Study Group XI, CHILL Language Definition, May 1980.

CONFERENCE PROCEEDINGS:

17. Dagless, E.L., "A Multimicroprocessor, CYBA-M", IFIP Congress 1977.

MEMORANDUM:

18. Barton, M.H.,"CHILL Performance Model CHPM, User's Guide." SERC DCS Annual Research Summary 1981, Report CYB110.

REFERENCE BOOK:

19. Reference Manual for the ADA Programming Language, US Department of Defense, July 1980.

JOURNAL ARTICLE:

20. Taylor, R., and Wilson, P., "Process-oriented language meets demands of distributed processing." Electronics, November 30, 1982.

CONTRACTOR IN-HOUSE REPORT:

21. Modelling of Interconnection Structure and Design of a Microprogram Development Environment, Final Report, RADC, Contract No. F49620-80-0012, June 1982.

UNPUBLISHED DISSERTATION:

22. Rubner, G.B., The Design and Implementation of a Sixteen-Bit Processor Node for CYBA-M", MSc Dissertation, University of Manchester, 1984.

CONFERENCE PROCEEDINGS:

23. Dagless, E.L., "A Multimicroprocessor: CYBA-M." Information Processing 77, B. Gilchrist (ed.), IFIP/North-Holland Publishing Co., 1977.

JOURNAL ARTICLE:

24. Burkimsher, P.C., "EMU: A multimicroprocessor software debugging tool." Software & Microsystems, 1, 2, 1982, pp. 41-47.

UNPUBLISHED DISSERTATION:

25. Kille, S.F., Modelling of Multiprocessor Interconnection Structures on CYBA-M", MSc Thesis, University of Manchester, September 1980.

CONTRACTOR IN-HOUSE REPORT:

26. Rivkin, L.S., Experiments concerning Buffer Insertion Loops, Department of Computation, UMIST, Progress Report, January 1982.

CONTRACTOR IN-HOUSE REPORT:

27. Barton, M.H., British Telecom-UMIST Research Fellowship, Document BTRF/15: Progress Report.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

END

FILMED

2-86

DTIC